

**A Mathematical Theory of Design:
Foundations, Algorithms and Applications**

Applied Optimization

Volume 17

Series Editors:

Panos M. Pardalos
University of Florida, U.S.A.

Donald Hearn
University of Florida, U.S.A.

The titles published in this series are listed at the end of this volume.

A Mathematical Theory of Design: Foundations, Algorithms and Applications

by

Dan Braha

*Department of Industrial Engineering,
Ben Gurion University,
Beer Sheva, Israel*

and

Oded Maimon

*Department of Industrial Engineering,
Tel-Aviv University,
Tel-Aviv, Israel*



SPRINGER-SCIENCE+BUSINESS MEDIA, B.V.

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN 978-1-4419-4798-7 ISBN 978-1-4757-2872-9 (eBook)
DOI 10.1007/978-1-4757-2872-9

Printed on acid-free paper

All Rights Reserved

© 1998 Springer Science+Business Media Dordrecht
Originally published by Kluwer Academic Publishers in 1998
Softcover reprint of the hardcover 1st edition 1998

No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

“the science of design is possible and some day we will be able to talk in terms of well-established theories and practices.”

Herbert Simon, 1969

To our families

CONTENTS

PREFACE	xix
Part One THE DESIGN PROCESS: PROPERTIES, PARADIGMS AND THE EVOLUTIONARY STRUCTURE	1
1 INTRODUCTION AND OVERVIEW	3
1.1 Scope and Objectives	3
1.2 Common Properties of Design	4
1.3 Design Theories	5
1.4 The Mathematical Method of General Design Systems	6
1.5 Difficulties of the Application of the Mathematical Method	8
1.6 Preview of the Book	9
1.6.1 Modeling the Attribute Space (Chapter 4)	11
1.6.2 The Idealized Design Process (Chapter 5)	11
1.6.3 The 'Real' Design Process (Chapter 6)	12
1.6.4 Computational Analysis of Design (Chapters 5-7)	13
1.6.5 The Measurement of A Design Structural and Functional Complexity (Chapters 8-9)	14
1.6.6 Algorithmic Methods and Design Applications (Parts III & IV)	14
1.7 Concluding Remarks	14
References	15
2 DESIGN AS SCIENTIFIC PROBLEM-SOLVING	19
2.1 Introduction	19
2.1.1 Motivation and Objectives	19
2.1.2 Overview of the Chapter	20
2.2 Properties of the Design Problem	22
2.2.1 The Ubiquity of Design	22
2.2.2 Design as A Purposeful Activity	23
2.2.3 Design is A Transformation Between Descriptions	23
2.2.4 Categories of Design Requirements	23
2.2.5 Bounded Rationality and Impreciseness of Design Problems	24
2.2.6 The Satisficing Nature of Design Problems	25

2.2.7	The Intractability of Design Problems	25
2.2.8	The Form of Design	25
2.3	Properties of the Design Process	27
2.3.1	Sequential and Iterative Natures of Design	27
2.3.2	The Evolutionary Nature of the Design Process	29
2.3.3	Design Process Categories	32
2.3.4	The Diagonalized Nature of Design	33
2.4	Survey of Design Paradigms	38
2.4.1	Defining A Design Paradigm	38
2.4.2	Design Paradigms	39
2.4.3	The Analysis-Synthesis-Evaluation (ASE) Design Paradigm	40
2.4.4	Case-Based Design Paradigm	44
2.4.5	The Cognitive Design Paradigm	45
2.4.6	The Creative Design Paradigm and the SIT Method	46
2.4.7	The Algorithmic Design Paradigm	57
2.4.8	The Artificial Intelligence Design Paradigm	58
2.4.9	Design as A Social Process	61
2.5	Scientific Study of Design Activities	65
2.5.1	The Axiomatic Theory of Design	66
2.5.2	Design as Scientific Problem-Solving	67
2.6	A General Design Methodology	77
2.7	Summary	79
	References	79
3	INTRODUCTORY CASE STUDIES	85
3.1	Electrical Design	85
3.1.1	Design of A "Data Tag"	85
3.1.2	Control Logic of Flexible Manufacturing Systems	90
3.1.3	Serial Binary Adder Unit Design	92
3.2	Mechanical Design	93
3.2.1	Mechanical Fasteners Design	93
3.2.2	Supercritical Fluid Chromatography (SFC) Design	96
3.2.3	Gear Box Design (Wormgear Reducer)	97
3.3	Flexible Manufacturing System Design	101
3.3.1	What is A Flexible Manufacturing System?	101
3.3.2	FMS Configuration Design Issues	102
3.4	Discussion	104
	References	106
Part Two	FORMAL DESIGN THEORY (FDT)	107
4	REPRESENTATION OF DESIGN ARTIFACTS	109
4.1	Introduction	109
4.2	Modeling the Artifact Space	111
4.2.1	The Basic Modeling of Design Artifacts	111

4.2.2	Examples	115
4.2.3	Properties of the Design Space	124
4.3	Summary	134
	Appendix - A (Proofs)	134
	Appendix - B (Basic Notions of Set Theory)	139
	References	141
5	THE IDEALIZED DESIGN PROCESS	143
5.1	Introduction	143
5.2	Motivating Scenarios	147
5.2.1	Mechanical Fasteners	147
5.2.2	The Car Horn	149
5.3	Preliminaries	152
5.3.1	The Attribute and Function Spaces	152
5.3.2	Proximity in Function and Attribute Spaces	154
5.3.3	Transformation Between Function and Attribute Spaces	160
5.3.4	Decomposition of Design Specification	163
5.3.5	Convergence of the Function Decomposition Stage	165
5.3.6	Order Relation for Attribute and Function Spaces	165
5.4	Idealized Design Process Axioms	167
5.4.1	Continuity in Function and Attribute Spaces	167
5.4.2	The Complexity of the “Homeomorphism” and “Continuity” Problems	171
5.5	Basis for Function and Attribute	172
5.5.1	Definition and Properties	172
5.5.2	Space Character as A Descriptive Complexity Measure	174
5.6	Concluding Remarks	175
	Appendix A - Basic Notions of Topology, and Language Theory	176
	Appendix B - Bounded Post Correspondence Problem (BPCP)	180
	Appendix C - Graph Isomorphism	180
	Appendix D - Proofs of Theorems	180
	References	185
6	MODELING THE EVOLUTIONARY DESIGN PROCESS	187
6.1	Introduction	187
6.2	Preview of the Models	190
6.2.1	Type-1 Design Process	190
6.2.2	Type-2 Design Process	197
6.3	Detailed Modeling	199
6.3.1	Type-0 Design Process $\langle L, Q, P, T_A, T_S, S_0, F \rangle$	200
6.3.2	Type-1 Design Process $\langle L, Q, P, T_A, T_S, S_0, F \rangle$	203
6.3.3	Type-2 Design Process $\langle L, Q, P, T, S_0, F \rangle$	203
6.4	Correctness and Complexity of the Design Process	205

6.4.1	Correctness of the Design Process	205
6.4.2	Computational Complexity of the Design Process Problem	207
6.5	Summary	214
	Appendix A - Basic Notions of Automata Theory [Adopted from 3]	215
	References	216
7	GUIDED HEURISTICS IN ENGINEERING DESIGN	217
7.1	Introduction	217
7.2	The Basic Synthesis Problem (BSP)	218
7.2.1	Problem Formulation	219
7.2.2	The Intractability of the BSP	222
7.3	The Constrained Basic Synthesis Problem (CBSP)	225
7.3.1	Problem Formulation	225
7.3.2	Universal Upper Bound on $ \mathcal{S} $	225
7.4	Refined Upper Bound on $ \mathcal{S} $	227
7.4.1	Probabilistic Design Selection	227
7.4.2	The Asymptotic Equipartition Property (AEP)	228
7.4.3	Consequences of the AEP on the CBSP	230
7.5	Design Heuristics for Feature Recognition	232
7.5.1	Geometric Modeling	232
7.5.2	Wireframe Feature Recognition	233
7.5.3	Combinatorial Analysis of the Connectivity Problem	235
7.5.4	Combinatorial Analysis of the Feature Recognition Problem	236
7.6	Summary	237
	Appendix A - The Satisfiability Problem	238
	References	238
8	THE MEASUREMENT OF A DESIGN STRUCTURAL AND FUNCTIONAL COMPLEXITY	241
8.1	Introduction	241
8.1.1	Complexity Judgment of Artifacts and Design Processes	241
8.1.2	Two Definitions of Design Complexity	243
8.1.3	Organization of the Chapter	245
8.2	Structural Design Complexity Measures	245
8.2.1	Description of the Valuation Measures	245
8.2.2	Basic Measures	247
8.2.3	Composite Measures	249
8.3	Evaluating the Total Assembly Time of A Product	255
8.3.1	Total Assembly Time and Assembly Time Measure	255
8.3.2	Assembly Defect Rates and Assembly Time Measure	261
8.3.3	Design Assembly Efficiency and Assembly Time Measure	262
8.4	Thermodynamics and the Design Process	267
8.4.1	Natural Science and Engineering Design	267
8.4.2	The "Balloon Model"	268

8.5	Functional Design Complexity Measure	273
8.6	Summary	276
	References	277
9	STATISTICAL ANALYSIS OF THE TIME COMPLEXITY MEASURE	279
9.1	Introduction	279
9.2	Other Methods for Design for Assembly (DFA)	280
9.3	Results and Discussion of the Time Complexity Measure	283
9.4	The Barkan and Hinckley Estimation Method	285
9.5	Conclusions	287
	Appendix A - Time Complexity Measure of A Motor Drive Assembly	289
	References	290
	Part Three ALGORITHMIC AND HEURISTIC METHODS FOR DESIGN DECISION SUPPORT	291
10	INTELLIGENT ADVISORY TOOL FOR DESIGN DECOMPOSITION	293
10.1	Introduction	293
10.2	AND/OR Tree Representation of Design	294
10.3	Guiding the AND/OR Search Tree	297
10.4	A Prototype System to Implement the Design Search Algorithm	300
10.4.1	Case-1 Overview	300
10.4.2	Basic Case-1 Definitions	302
10.4.3	The Case Builder Interface	305
10.4.4	The Analyzer Interface	312
10.5	Summary	318
	References	319
11	PHYSICAL DESIGN OF PRINTED CIRCUIT BOARDS: GROUP TECHNOLOGY APPROACH	321
11.1	Introduction	321
11.1.1	The Role of Clustering (Grouping) in Design	321
11.1.2	The Circuit Partitioning Problem	324
11.2	Mathematical Formulation	330
11.3	Properties of the Circuit-Partitioning Problem	332
11.4	A Grouping Heuristic for the Circuit-Partitioning Problem	334
11.5	A Branch and Bound Algorithm	336
11.6	Computational Results Using the Branch and Bound Algorithm	339
11.7	Summary	345
	Appendix A - A Brief Overview of Microelectronics Circuits and their Design	345
	Appendix B - (Proof of Theorem 11.1)	348
	Appendix C - (Bounds on the Number of Times Net Type i Must be	349

Packed)	
References	351
12 PHYSICAL DESIGN OF PRINTED CIRCUIT BOARDS: GENETIC ALGORITHM APPROACH	353
12.1 Introduction	353
12.2 The Genetic Algorithm Approach	354
12.3 A Genetic Algorithm for the Circuit-Partitioning Problem	356
12.4 Computational Results	358
12.5 Other Applications of Genetic Algorithm	361
12.5.1 The Catalogue Selection Problem	361
12.5.2 Outline of the Genetic Algorithm	362
12.6 Summary	363
References	363
13 ADAPTIVE LEARNING FOR SUCCESSFUL DESIGN	365
13.1 Introduction	365
13.1.1 Managing the Intricate Correspondence Between Function and Structure	365
13.1.2 The Applicability of the Methodology	367
13.2 Problem Formulation	368
13.3 Adaptive Learning of Successful Design	369
13.3.1 The Probabilistic Nature of the Design Process	369
13.3.2 Preliminaries	371
13.3.3 The P-Learning Algorithm	374
13.4 Illustrative Example	376
13.5 A Catalogue Structure for the P-Learning Algorithm	381
13.6 Summary	382
Appendix A - Computation of the Experimental Success Probabilities	383
Appendix B - Bayes' Theorem	384
References	385
14 MAINTAINING CONSISTENCY IN THE DESIGN PROCESS	387
14.1 Introduction	387
14.1.1 Variational Design	387
14.1.2 Design Consistency in Variational Design	388
14.1.3 Chapter Outline	389
14.2 Previous Efforts	389
14.2.1 Geometric Reasoning	389
14.2.2 Numerical Techniques based on Continuation Methods	390
14.2.3 Other Numerical Techniques	391
14.2.4 Discussion	393
14.3 Design Consistency	394

14.4 Design Evolution in Variational Design Systems	396
14.5 Design Consistency Through Solution Trajectories	400
14.5.1 Definitions in Design Consistency	400
14.5.2 Theorems in Design Consistency	403
14.6 COAST Algorithm for Design Consistency	405
14.6.1 Mean Value Theorem	405
14.6.2 Rigorous Sensitivity Analysis Algorithm	406
14.6.3 Bifurcations and Infeasible Regions	408
14.7 Design of Cantilever Beam	409
14.7.1 Design Execution	410
14.7.2 Comparison with Other Methods	414
14.8 Summary	416
Appendix A - Interval Analysis Techniques	417
A.1 Solving Systems of Interval Equations	418
A.2 Existence and Uniqueness of Solutions	419
Appendix B - Constraint Model of Beam	419
References	421
15 CONSTRAINT-BASED DESIGN OF FAIRED PARAMETRIC CURVES	423
15.1 Constraint-Based Curve Design	423
15.2 Previous Work	424
15.3 Maintaining Design Consistency in Constraint-Based Curve Design	425
15.3.1 Distance Constraints	426
15.3.2 Arc Length	428
15.3.3 Consistency in Curve Fairing	428
15.3.4 COAST Methodology for Design Consistency	430
15.4 Examples	431
15.4.1 Bezier Curve from Distance Constraints	431
15.4.2 Apparel Design	435
15.5 Discussion	443
References	443
16 CREATING A CONSISTENT 3-D VIRTUAL LAST FOR PROBLEMS IN THE SHOE INDUSTRY	445
16.1 Problems in Shoe Design Industry	445
16.2 Creation of A Virtual Last	448
16.2.1 Constraint Definitions	452
16.2.2 Curve Fairing	455
16.3 Results	456
16.4 Summary	459
References	459

Part Four DETAILED DESIGN APPLICATIONS	461
17 DESIGN OF A WORMGEAR REDUCER: A CASE STUDY	463
17.1 Introduction	463
17.2 Conceptual Design of A Wormgear Reducer (Gear Box)	464
17.2.1 Confrontation	464
17.2.2 Problem Formulation	465
17.2.3 Design Concepts	466
17.3 Detailed Synthesis of the Gear Box	468
17.3.1 Motor Design	470
17.3.2 The Design of the Transmission Parts and the Outline of Their Relative Position	471
17.3.3 Testing the Current Design Against the Wormgear Load and Strength Constraints	475
17.3.4 Initial Design of the Casing (Box)	477
17.3.5 The Design of the Wormgear Shaft Set	479
17.3.6 Calculation and Check of the Shaft Set Parts	481
17.3.7 Strength and Wear-Resistance Constraints	486
17.3.8 Detailed Design of the Casing	487
17.3.9 Accessories Design	487
17.3.10 Casing Heat Balance Constraints	488
17.4 Discussion	491
17.4.1 Design Description (L)	493
17.4.2 Transformation (T)	494
17.5 A Methodology for Variational Design	495
17.5.1 The General Methodology	495
17.5.2 Demonstration	496
17.5.3 Design Execution	498
References	498
18 ADAPTIVE LEARNING FOR SUCCESSFUL FLEXIBLE MANUFACTURING CELL DESIGN: A CASE STUDY	499
18.1 Introduction	499
18.2 Physical Configuration	500
18.3 Parameters and Performance Measures	503
18.3.1 Performance Measures	503
18.3.2 Parameters and Structural Assumptions	504
18.3.3 Evaluation of the Responses Through Simulation	506
18.4 Solving the Design Problem Using the P-Learning Algorithm	506
18.5 Concluding Remarks	512
References	512

19 MAINTAINING DESIGN CONSISTENCY: EXAMPLES	513
19.1 Wormgear Assembly Problem	514
19.1.1 Dimensions - Wormgear Assembly	514
19.1.2 Design Execution	514
19.2 Helical Compression Spring Problem	521
19.3 Other Design Areas	524
19.4 Point at A Distance from Two Points	528
19.5 Line Tangent to Two Circles	533
19.6 Helical Compression Spring (Continued)	539
Appendix A - Constraint Model of Wormgear	548
20 CASES IN EVOLUTIONARY DESIGN PROCESSES	551
20.1 Automobile Design Example	551
20.1.1 The Specification and Design Description Properties	551
20.1.2 The Production Rules	553
20.1.3 Car Synthesis Using the Design Search Algorithm (see Chapter 10.3)	562
20.2 Forklift Design Example	564
20.2.1 The Specification and Design Description Properties	565
20.2.2 The Production Rules	568
20.2.3 Forklift Truck Synthesis Using the Design Search Algorithm (see Chapter 10.3)	576
20.3 Computer Classroom Design Example	578
20.3.1 The Specification and Design Description Properties	578
20.3.2 The Production Rules	584
20.3.3 Computer Classroom Synthesis Using the Design Search Algorithm (see Chapter 10.3)	601
20.4 Tire Design Example	604
20.4.1 The Specification and Design Description Properties	606
20.4.2 The Production Rules	608
20.4.3 Tire Synthesis Using the Design Search Algorithm (see Chapter 10.3)	612
20.5 Fastener Design Example	614
20.5.1 The Specification and Design Description Properties	614
20.5.2 The Production Rules	615
20.5.3 Fastener Synthesis Using the Design Search Algorithm (see Chapter 10.3)	620
20.6 Fastener Design Example (Continued)	621
20.6.1 The Specification and Design Description Properties	621
20.6.2 The Production Rules	624
Appendix A - Automobile Design	632
A.1 Engines	632
A.2 The Body	638
A.3 Steering System	639
A.4 Suspension System	640

A.5 The Brake System	642
A.6 Power Train	644
A.7 Other Design Consideration	645
Part Five PRACTICAL CONSIDERATIONS	649
21 CONCLUDING REFLECTIONS	651
21.1 General Purpose Guidelines	651
21.1.1 Representation of Design Knowledge	651
21.1.2 Design Process	652
21.2 Algorithmic Design Guidelines	662
21.2.1 Logic Decomposition and Case Based Reasoning (Chapter 10)	662
21.2.2 Group Technology and Clustering Analysis (Chapter 11)	665
21.2.3 Solving Design Problems with Genetic Algorithms (Chapter 12)	666
21.2.4 Probabilistic Selection Methods for System Design (Chapter 13)	667
21.2.5 Maintaining Consistency in the Design Process (Chapters 14, 15, 16)	667
21.3 Summary	670
REFERENCES	670
INDEX	671

PREFACE

Formal Design Theory (FDT) is a mathematical theory of design. The main goal of FDT is to develop a domain independent core model of the design process. The book focuses the reader's attention on the process by which ideas originate and are developed into workable products. In developing FDT, we have been striving toward what has been expressed by the distinguished scholar Simon (1969): that "the science of design is possible and some day we will be able to talk in terms of well-established theories and practices."

The book is divided into five interrelated parts. The conceptual approach is presented first (Part I); followed by the theoretical foundations of FDT (Part II), and from which the algorithmic and pragmatic implications are deduced (Part III). Finally, detailed case-studies illustrate the theory and the methods of the design process (Part IV), and additional practical considerations are evaluated (Part V). The generic nature of the concepts, theory and methods are validated by examples from a variety of disciplines.

FDT explores issues such as: algebraic representation of design artifacts, idealized design process cycle, and computational analysis and measurement of design process complexity and quality.

FDT's axioms convey the assumptions of the theory about the nature of artifacts, and potential modifications of the artifacts in achieving desired goals or functionality. By being able to state these axioms explicitly, it is possible to derive theorems and corollaries, as well as to develop specific analytical and constructive methodologies.

The desired design function and constraints are mapped to the artifact description using an evolutionary process that can be viewed as a feedback loop of double interleaved automata accountable for both analysis and synthesis activities. The automata modify the specification world and the design artifact until a solution is achieved. Inherent properties, such as soundness and completeness of the process, are also explored and proved.

A special case of the synthesis activity, called the Basic Synthesis Problem (BSP), is shown to be intractable (NP-Complete). We show that by combining domain-specific mechanical engineering heuristics, which constrain the structure of potential artifacts, the BSP will be computationally tractable. In order to extend this guided heuristics approach to other domains, the heuristically guided combinatorial analysis will be presented for 2-D wireframe feature recognition systems which are predominant in industrial CAD systems.

Information theory is utilized to quantitatively articulate two definitions of design complexity (*structural* complexity versus *functional* complexity), their

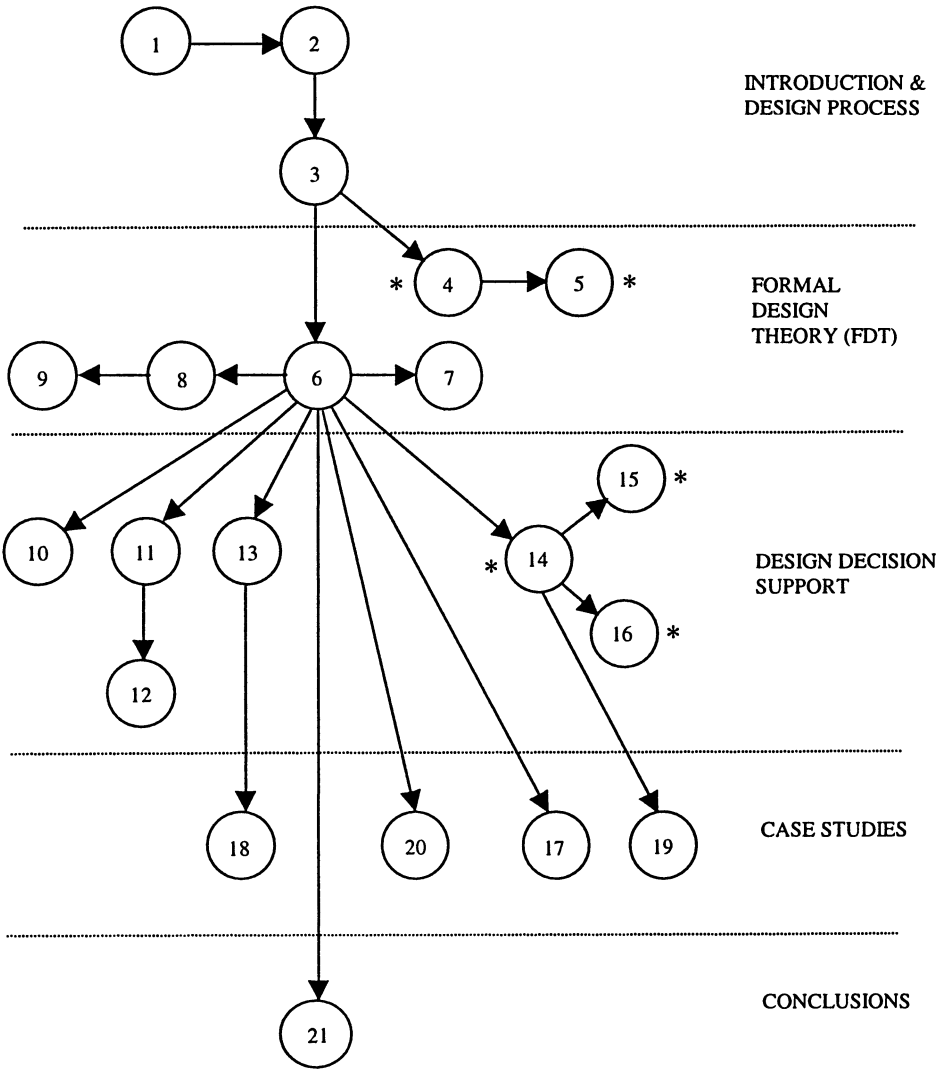
associated value measures, and the relationships between them. The proposed measurable metrics provide a proper basis for evaluating each step in the design process hierarchy, and accordingly recommend the direction to follow for design modification and enhancement. It also provides a framework for comparing competing artifacts (the output of a design process). In addition, we demonstrate that product assembly time can be predicted by applying the design complexity measures. It is found that the correlation between the proposed assembly time equation and the estimation of the assembly time as derived from Boothroyd and Dewhurst' Design for Assembly (DFA) methodology is very close to +1 over a wide diversity of experiments.

Based on the theory, algorithms and heuristic methods are developed in Part III. Methods include case based reasoning, group technology, genetic algorithms, adaptive learning and convergent probabilistic search for successful design, and continuation methods for maintaining design consistency. The theory is also the basis for a computer CADAT (Case-based Design Advisory Tool) program. CADAT suggests the specification decomposition steps to follow in each design cycle. It is a combined algebraic-expert system program, which explores the evolutionary nature of the design process.

In Part IV we present detailed design applications (e.g. the design of wormgear reducer, helical compression spring, automobile, apparel, shoes, and flexible manufacturing system) that utilize the processes and methods developed in this book.

The book is intended for theoreticians and practitioners in system and engineering design. It can also serve as the basis of an upper level undergraduate or graduate course in computer science or a variety of engineering disciplines (such as mechanical, electrical, industrial, chemical and civil engineering). The practitioners will have algorithmic references for the design process, validated and demonstrated by detailed real world case studies. The book will also provide a unified and explanatory model for the design process. In addition, engineers, computer scientists and programmers; who are involved in design software, will find herein the theoretical foundations and detailed algorithms for the next generations of Computer Aided Design (CAD) software. However, the book will not dwell on the fundamentals of integrated circuitry, solid mechanics, fluid mechanics, and the like, since they are more than adequately covered in the regular, domain-specific engineering design literature.

It is recommended to read the introductory Chapters 1-3 first. Prerequisite dependencies between individual chapters of the book are illustrated in Figure 0.1. For example, an arrow from the number 3 to 6 indicates that it is recommended that Chapter 3 be read before Chapter 6. Although Chapter 21 can be read independently of other chapters, we suggest reading it for "dessert." We have used asterisks in Figure 0.1 to note the use of more powerful mathematical tools than are required in the rest of the book. The following system of numbering and cross-referencing is used in the book. Items, such as sections, definitions, theorems, examples, remarks, etc., are numbered consecutively within each chapter. Section numbers in the text are in italic type. Cross-references are in the form "by Definition 3.4"; this means "by Definition 4 of Chapter 3."



* advanced students and researchers

Figure 0.1 Prerequisite Dependencies of the Book

Acknowledgements

We acknowledge Irad Ben-Gal, Michael Caramanis, Leonid Charny, Li Keping and Vince Huffaker of Boston University; Roni Horowitz of Tel-Aviv University; Armin Schmilovici and Kenneth Preiss of Ben-Gurion University; and Sageet Braha for their helpful comments and practical discussions. The useful suggestions made by our students are also appreciatively acknowledged. Finally, we are grateful for the thoughtful support of our families.

NOTE: Authors' names have been listed in alphabetical order.

PART ONE

***THE DESIGN PROCESS: PROPERTIES,
PARADIGMS AND THE EVOLUTIONARY
STRUCTURE***

CHAPTER 1

INTRODUCTION AND OVERVIEW

This book presents Formal Design Theory (FDT), a mathematical theory of design. The main goal of FDT is to develop a domain independent core model of the design process. FDT explores issues such as: the algebraic representation of design artifacts, idealized design process cycle, and computational analysis and measurement of design process complexity and quality. FDT can be used as a framework for the future development of automated design systems (adding another dimension to the current CAD systems).

1.1 SCOPE AND OBJECTIVES

Design, as problem solving, is a natural and ubiquitous human activity. Design begins with the acknowledgment of needs and dissatisfaction with how things stand, and hence the realization that some action must take place in order to solve the problem. As such, the issue of design is of major concern to all disciplines within the artificial sciences (engineering in the broad sense). Independent of the specific design domain; we claim that design problems share a core of common properties. Thus making it reasonable to talk in terms of well-established general design theory and practices.

The purpose of this book is to offer a new and integrative, mathematically formulated, approach to the study of formal models in design sciences. Formal Design Theory (FDT) is an area within the mathematical-natural sciences that is still in the exploratory stage of development. It is concerned primarily with design artifacts and their processes. As an experimental and formal science, it deals only with those properties that remain invariant under transformation from one design domain to another. The theory is already being utilized in various ways; such as, identifying the appropriate links between the creativity process and human design activity, and developing systems that interface with existing CAD systems in order to identify successful design configurations.

The investigation that led to the discovery of this branch of mathematical science was motivated by the recognized need for a substantial foundation in theoretical and experimental science by Design Science in general, and Design Engineering in particular. To some extent this need may have been met in part [43, 19, 20, 3, 37, 44, 51], by the mathematical derivation of relationships and methods that appear to govern the design processes with respect to the:

1. Design process framework;
2. Nature of an evolutionary scheme;
3. Idealized design process cycle;
4. Computational complexity analysis;
5. Complexity measures, such as the number of mental discriminations and time taken to derive a design solution;
6. Logic decomposition and case based reasoning;
7. Group technology approach;
8. Genetic algorithms;
9. Adaptive learning for successful design;
10. Consistent configuration in iterative design.

It should also be mentioned that the mathematical approach used must rely upon the experimental validation of relationships in addition to the proof of theorems. Consequently, as in any branch of natural science, any theory developed or laws discovered may only be valid within the domain in which they have been tested.

Our vision in developing FDT was to achieve what was expressed by the renowned scientist Simon (1969); that the science of design is possible and some day we will be able to talk in terms of well-established theories and practices. Proclus' aphorism that "it is necessary to know beforehand what is sought" suggests a ground rule for recognizing properly any new field of study. A new field needs to scrutinize its bounds and objectives, where it stands in the universe, and how it proposes to relate to the established disciplines. Such clarification is the objective of this first chapter.

1.2 COMMON PROPERTIES OF DESIGN

Various engineering related disciplines (including mechanical, electrical, industrial, and chemical engineering; as well as computer science and management science) deal with certain stages in the design process. Each discipline has developed its own paradigms, models, and semantics; which overlap, differ, and sometimes contradict. Design has been discussed in contexts such as general design methodologies [30, 16, 31, 11, 1, 5, 6]; design artifact representation [8, 14, 26, 35, 25, 22]; computational models for the design process [21, 23, 24, 27, 34, 18]; and knowledge-based CAD systems [9, 33, 28]. Based on the design research, we observe that independent of the specific design domain, design problems share a core of common properties that can either be logically derived or empirically verified. Let us review some of these basic features as related to the design process:

- Design begins with an acknowledgment of needs that highlight discrepancies with the current status. Thus revealing that some action must take place in order to bridge the gap.
- Designing an artifact can be considered a transition from concepts and ideas to concrete descriptions. In the case of engineering design, such design descriptions

range from specifications in formal languages (such as computer-aided engineering systems, symbolic programming techniques associated with AI, and hardware design/description languages); to descriptions in quasi-formal notation (such as linguistic descriptions and qualitative influence graphs); and to very informal and visual descriptions (such as functional block diagrams, flow-diagrams, and engineering drawings).

- The designer is constantly faced with the problem of bounded rationality. In the model of bounded rationality it is self-evident that there are limitations on the cognitive and information processing capabilities of the designer's decision making.
- The design specifications, which constitute the constraints of a design problem, may initially not be precise or complete; hence, the evolution and elaboration of specifications becomes an integral part of the design process [6].
- Traditional engineering design methods make much more use of satisfactory criteria (bounded rationality) rather than optimal (pure rationality) specifications.
- Alternatives and design solutions are not provided in advance or in a constructive manner. They must be found, developed, and synthesized by some research process; which is iterative and evolutionary in nature.

We make explicit use of the above properties in subsequent sections where the formal theory of design is presented.

1.3 DESIGN THEORIES

Over the last decades, a significant amount of research has been devoted to developing design theories [12, 5, 16, 10, 31, 15, 13, 45, 47]. These theories have been classified into various conceptual categories such as empirical, descriptive, or prescriptive [40]; as well as into their geographic origin [36, 39, 41, 46]. FDT is based on mathematical foundations as opposed to axioms and corollaries that are not based on a formal system; e.g., [50, 32].

The field of design theory is relatively new. It has been particularly stimulated by three computer-related technological advances: computer-aided design (CAD) [2], knowledge-based expert systems (KBES) [33], and concurrent engineering (CE) [29]. The slow development and confusion about design theory can be mainly attributed to the lack of sufficient scientific foundation in engineering design. Dixon [7] has argued that engineering design education and practice lack an adequate base of scientific principles, and are guided too much by specialized empiricism, intuition, and experience. Kuhn [17] concluded that design is at a prescience phase and must go through several phases before it constitutes a mature science (hence theory). A mature science is the state of a discipline in which there is a coherent tradition of scientific research and practice embodying law, theory, application, and instrumentation. In order to achieve this kind of maturity, design researchers need to borrow methodologies from other disciplines (beyond calculus) that have reached relative scientific maturity [4].

A design method, within a design paradigm, does not constitute a theory. Theory emerges when there is a testable explanation of why the method behaves as it does [7]. Design methods do not attempt to say what design is, or how human designers do what they do, but rather provide tools by which designers can explain and perhaps even replicate certain aspects of design behaviors. The major components and aims of design theories, which our book is also committed to, are:

1. To construct a systematic inquiry into the design process, and to uncover some intelligible structure or pattern underlying the phenomenon. In other words, a theory of design must be in part descriptive;
2. To specify how design should be done, and to allow the construction of more rational methods and tools to support practical design. In other words, a theory of design must be in part prescriptive.
3. To construct a simple design theory: When two design theories are possible, we provisionally choose the one which our minds judge to be the simpler, on the supposition that this is the more likely to lead in the direction of the truth. This simplicity principle includes as a special case the principle of William of Occam "Causes shall not be multiplied beyond necessity";
4. To construct a consistent design theory: A design theory must be consistent with whatever else we know or believe to be true about the universe in which the phenomenon is observed;
5. To develop a general design theory: The value of a design theory will be determined, to a great extent, by its generality and domain-independence (including mechanical, industrial, electrical, and civil engineering; as well as computer science).

1.4 THE MATHEMATICAL METHOD OF GENERAL DESIGN SYSTEMS

One of the most interesting hypothesis that emanates from the significant amount of design research is that in all the diverse richness of design, and regardless of the various methodologies or models about the structure of the design process, one may identify a single general theory. The theory that design as a problem solving activity conforms to one of the most powerful and successful human activities; namely, scientific discovery. Examining the literature on design theory [12, 5, 16, 10, 31, 15, 13, 45, 47] reveals a number of intermittent and closely intertwined issues (that are resolved to some degree): Can we construct a natural logic of design? What is the connection between design and science? How can designs be synthesized computationally? Unfortunately, despite the numerous design theories and the metaphor of design-as-scientific discovery; there is no common core of mathematical general design theory. Consequently, we feel that the most interesting themes in design theory are: What is the nature of design knowledge representation? Should we try to formalize the design process? What is the connection between general design and mathematics? We hope to draw the reader's attention to mathematical methods,

which diverge considerably from the techniques applied by contemporary designers; and to shed some further perspective on the structure of design processes.

While much of what will be discussed applies to design disciplines in general, our particular emphasis will be the design of well-structured design systems that assumes the existence of a formal language for representing the structural and functional properties of a device. We believe that a formal model of design can help in developing a generally accepted core of mathematical design theory, which could be applied to enhancing the conceptual design stage. The theory can also provide a reference framework within which the potentialities and limitations, as well as the efficacy and deficiency of design automation systems (e.g., computer-aided design (CAD) and "expert" design systems) can be evaluated.

The philosophical view that underlies the theory is the constructivist or systems approach. Klir [42] defines systems science as "a science whose domain of inquiry consists of those properties of systems and associated problems which emanate from the relation-oriented classification of systems." The general study of "thinghood" is essentially physics; hence, it follows that the study of any system is subsumed in its physics (e.g., the underlying fluid mechanics, heat transfer, phase transformation, and solid mechanics of an arc welding design). This is the essence of reductionism (see Section 1.5.2). On the other hand, we find the antithetic claim that all apparent properties of "thinghood" are already subsumed under "systemhood." Essentially, this is holism. The way in which this holistic view is used in this book is quite different. We see reductionism and holism as complementary, or cognitively adjoined. For the description of the design systems in which we are interested; reductionism implies attention to a lower level, while holism implies attention to a higher level. These are intertwined in any satisfactory description, and each entails some loss and some gain relative to our cognitive preferences.

Designers are more or less familiar with the construction of mathematical representations or models of engineering design (the use of calculus). However, it is perhaps not widely recognized (among designers) that mathematicians often construct other mathematical systems. The theory presented here is grounded, at large, in those branches of mathematics called Information Theory, Logic, Automata and Complexity Theory, Generalized Topology and the Theory of Category.

Topology is the branch of mathematics that studies the notion of continuity. The systems studied in topology are those mathematical structures (called topological spaces) on which continuous processes can be defined. One of the basic problems of topology is to determine whether two such spaces are homeomorphic. Homeomorphism means that such spaces are indistinguishable from one another from the standpoint of the continuous processes that may be defined on these spaces. Category Theory is a general theory for modeling the relations that exist between formal systems. Thus, for example, we identify the attribute space (alternately the artifact space, structure space, or the design space) as an algebraic structure; the relations between the function and attribute spaces as a homeomorphism; and the evolutionary structure of the design process as consisting of a specification language expressed by first-order logic and two generative mechanisms for decomposing specifications and generating new structural attributes.

1.5 DIFFICULTIES OF THE APPLICATION OF THE MATHEMATICAL METHOD

Let us briefly discuss the role mathematics may play in the development of general design theory. At present, there exists no universal system of mathematical design theory. We believe that if one is ever developed, it will probably not be during the next several decades. Considering the fact that design theory is much more complex, far less understood, and undoubtedly significantly newer than many natural sciences; one should expect design theory to lag considerably behind (in terms of mathematical formalization) fields such as physics.

The assumption that mathematics can not be applied to design theory seems to be based on a variety of conceptions including:

- That the human element, psychological factors, etc. dictate design.
- That there are no quantitative measures for important factors in design.
- That a mathematical method would rule out the possibility of creativity in design.

These conceptions may be undermined by noticing that the outlook on the physical and biological sciences during their early periods was similarly pessimistic. In addition, to continue the simile with physics, mathematical theory was necessary before precise measurements of the quantity and quality of heat (energy and temperature) could be obtained. Thus, we must look beyond the above conceptions in order to explain why mathematics has not been rigorously applied to design theory.

To begin with, design problems have not been formulated clearly and have often been stated in vague terms (such as artifact, adaptiveness, etc.). These factors, a priori, make mathematical reasoning appear hopeless. Even in design theories where a description problem has been handled more satisfactorily, mathematical tools have seldom been used appropriately. Next, the empirical background for design theory (e.g., cognitive studies) is inadequate. Indeed, it would have been absurd in physics to anticipate Kepler and Newton theories without Tycho de Brahe's astronomical observations. There is no reason to hope that the development of design theory would be easier.

In light of these remarks we may describe our own position as follows: The aim of this book does not lie in the direction of empirical research. This task seems to transcend the limits of any individually planned program. We shall attempt to utilize some commonplace design experiences concerning simple devices and designer behavior that lend themselves to mathematical treatment and which is of importance to design. It should be added that the treatment of these manageable situations may lead to results that are already known, but the exact mathematical proofs may nevertheless have been lacking. The theories do not exist as scientific theories until proofs have been given. For example, the data for the orbit of Mars was available long before its course had been derived by Kepler's elliptical law. Similarly, in design theory, certain results (such as hierarchy and the intractability of design problems) may be already known. Yet, it is of interest to derive them again from an exact

theory. Finally, we utilize comprehensible design examples in order to better focus on the fundamental governing principles. The free fall is a very basic physical phenomenon, which brought forth mechanics. Designers often tend to focus on “pressing” practical issues (such as providing a needed artifact), and tend to set aside issues (such as formalization) that do not bring them to immediate applications. This orientation, as indicated by other sciences, merely delays progress (including “burning” questions). It seems to us that the same research plan should be applied in design.

1.6 PREVIEW OF THE BOOK

A general model of design can be visualized as consisting of the following tasks: needs assessment, analysis, decomposition, synthesis, and evaluation. The first task is concerned with the *assessment* of the desired needs and requirements, which are often fuzzy in nature. *Analysis* involves specifying, identifying, and preparing the problem as well as developing an explicit statement of goals. *Decomposition* is concerned with breaking the problem into parts and defining the boundaries of a space in which a fruitful search for the solution can take place. *Synthesis* is concerned with discovering the consequences of putting the new arrangement into practice. *Evaluation* is concerned with judging the validity of the solutions relative to the goals and selecting among alternatives. This implies that the outcome from the evaluation phase is revised and improved by reexamining the analysis (inner cycle). The outcome may also revise the perceived needs (outer cycle).

In Chapter 2, we examine the logic and methodology of engineering design from the perspective of the philosophy of science. The fundamental characteristics of design problems and design processes are discussed and analyzed. These characteristics establish the framework within which different design paradigms are examined. Following the discussions on descriptive properties of design, and the prescriptive role of design paradigms, we advocate the plausible hypothesis that there is a direct resemblance between the structure of design processes and the problem solving of scientific communities.

In Chapter 3, we provide some examples of design processes. These cases were chosen to give the reader insight into the design process (emphasizing the synthesis part), and the thinking of designers when involved with five different design situations.

Formal Design Theory (FDT) is established in Part II (Chapters 4-9). FDT is a descriptive model that attempts to explain how design artifacts are conceptually represented; and how design processes are conceptually performed (as captured by the Analysis, Decomposition, and Synthesis stages) in terms of knowledge manipulation.

Prerequisite dependencies between individual components of FDT are expressed by the diagram in Figure 1.1.

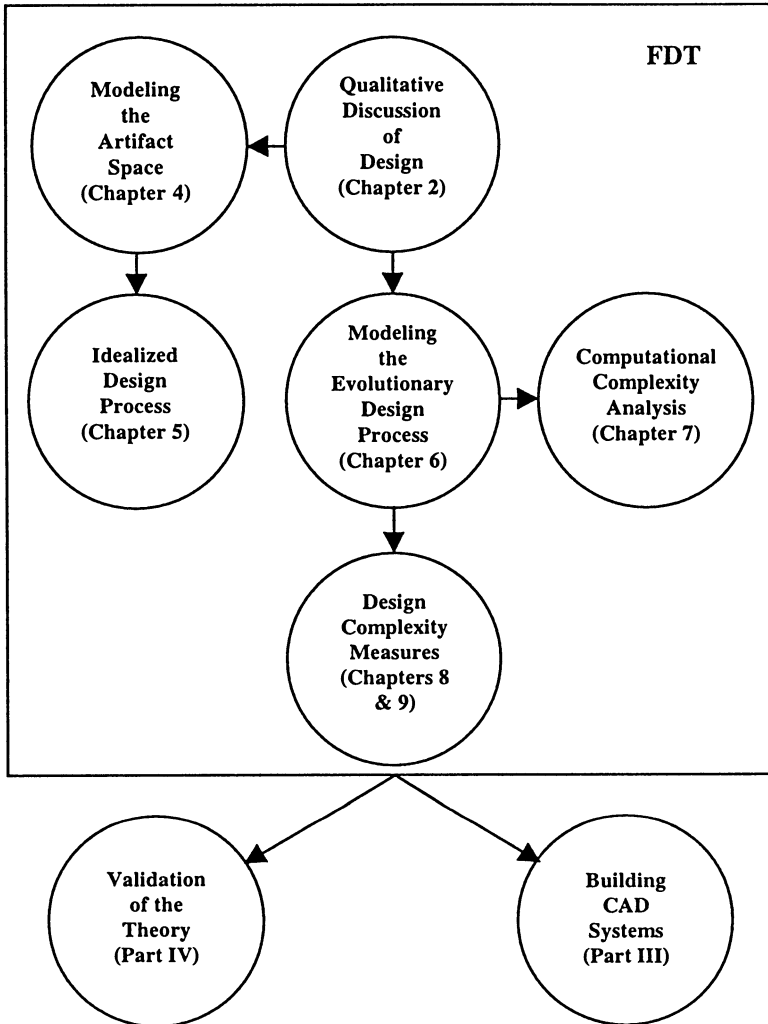


Figure 1.1 Prerequisite Dependencies of FDT

FDT has inherent relationships with other fields. FDT intersects with mathematics (generalized topology and category theory), computer science (complexity theory, automata and formal logic), physics (thermodynamics and statistical mechanics), information theory (the asymptotic equipartition property) and psychology (categorical judgments). Part II describes these relationships.

1.6.1 MODELING THE ATTRIBUTE SPACE (CHAPTER 4)

Chapter 4 lays out a domain independent modeling of design artifacts, which is based on the following postulates:

Postulate 1 (Entity-Relational Knowledge Representation): an artifact representation is built upon the multiplicity of modules (attributes) and relationships among them.

Postulate 1 supports the stage of conceptual design; which includes the assessment, analysis, and decomposition tasks. The conceptual design stage has a great impact on the direction of the design project; including its performance, manufacturability, production cost and other concurrent engineering factors. During this stage, continuous properties are usually discretized. The discreteness might cause the design evaluation process to be imprecise. However, in most situations the aim of the conceptual design stage is to generate a variety of alternatives that meet a required set of specifications early in the project rather than to produce detailed engineering drawings that are describable by continuous properties.

Postulate 2 (Nested Hierarchical Representation): the design of any complex system can be considered at various abstraction levels. The general direction of design is from more abstract to less. A design at any level of abstraction is a description of an organized collection of constraints (such as various structural, cause-effect, functional, and performance features) that appear in the physically implemented design.

These premises lead directly to formulating the attribute space as an *algebraic structure*. The artifact is represented by the pair $\langle M, C \rangle$. M stands for the set of modules that the artifact is comprised of; and C denotes the set of relations that represent the relationships among the modules. In order to capture the essence of design, a hierarchical construction of systems from subsystems is also developed. Consequently, the general set of modules is classified into *basic* and *complex modules*. Basic modules represent entities that can not be defined in terms of others. Complex modules are defined hierarchically in terms of other modules, where the effects of their interactions are expressed.

Any prescriptive approach to design should have some ability to forward search (from initial components toward the specifications) and backward search (from initial specifications toward the initial components). To conduct these search strategies, Chapter 4 rigorously defines three types of design-space operators; and explores their characteristics. These operators are able to manipulate transitive relationships among several relations, and perform any search tasks on a given attribute space. Two central types of operators are defined: *Composition* (undertake composition tasks) and *Decomposition* (undertake decomposition tasks). *Integration* operators combine both.

1.6.2 THE IDEALIZED DESIGN PROCESS (CHAPTER 5)

In FDT, the idealized design process is defined as a mapping of the desired set of

specifications (describing the desired functions and constraints of the final product) to the *artifact description* (the final detailed product description). Ideally, a design process includes several steps: (1) in order to provide a rule of direct correspondence between structural and functional attributes, the attributes included in the *artifact space* (which includes representations of artifacts in terms of properties, drawings, etc.) are mapped to functional properties that form the *function space* (represented in terms of functional descriptions, and artifact behavior). This step conceptualizes the issue of design *analysis*; (2) the desired set of initial specifications (describing the desired functions and constraints of the final product) is transformed to a “close” functional description that is sufficiently detailed. This “close” functional description represents the concept of a *model* as mediating between the function and attribute spaces; (3) finally, the detailed functional description of the artifact is mapped to the attributes or the structure of the artifact (part of the attribute space) by the “inverse” of the analysis mapping (i.e., *synthesis*). These attributes are observable properties that are needed in order to manufacture the artifact. For example, if we consider mechanical machines, the functional specifications are transformed to a graphic representation of the machine (e.g., a dynamic model of its behavior); which is transformed to the physical description of the machine (described in terms of part types, attributes, and dimensions).

In principle, these three processes (analysis, “close” functional description, and synthesis) can be combined in a cyclic operation. Since neither the “close” functional description of the artifact nor the selection process (i.e., synthesis) can be rigorous and practical simultaneously, a loss of information is to be expected. Consequently, the described cycle is considered idealized in the same sense as in thermodynamics; where a frictionless (hence ideal) heat engine follows the cycle of Carnot.

Thus, the transformation between the function space and the artifact space must consist of some combination of processes that make up the idealized cycle. We characterize the idealized cycle by a set of properties (emphasizing the *continuity* property), which immanently embody any idealized design process. These properties reflect the loss of information occurring in the course of the cycle. The overall exploration is performed by applying the methodology of generalized topology.

We introduce the concept of a *basis* for a function space (or attribute space). The intuitive meaning of a *basis* for a function space (or attribute space) is the collection of already known and decomposable functions (or attributes) that can be utilized to decompose a new function. Thus, the designer can utilize the basis functions (attributes), which have already been solved, to more easily solve a new problem.

1.6.3 THE 'REAL' DESIGN PROCESS (CHAPTER 6)

In the design process as in many fields, real knowledge, unlike the ideal knowledge, is fuzzy and one must take into consideration the following characteristics:

- The idealized design process is regarded as a direct mapping from the functional space onto the attribute space, while the real design process is regarded as a

stepwise refinement and evolutionary transformation procedure where the designer seeks the solution that can satisfy requirements;

- The real design process uses of the artifact's behavior instead of its functional requirements;
- The real design process takes into account the physical constraints, which were ignored in the presence of ideal knowledge.

We view the real design process as a goal-directed derivation process, which starts with an initial specifications and terminates with one or more artifact descriptions. By adaptively modifying pairs of artifact descriptions and specifications, from one step to the next, we arrive at a design solution. The desired functional requirements and constraints are mapped to the artifact description using an evolutionary process that can be viewed as a feedback loop of double interleaved automata accountable for both analysis and synthesis activities. The automata modify the specifications and the design artifact until a solution is reached. Properties such as *soundness* and *completeness* of the process are explored. The real design process model is based on the following postulates:

Postulate 3 (Incompleteness): Any knowledge representation (as presented by the designer) is incomplete.

Postulate 4 (Bounded Rationality): The designer can consider only a subset of knowledge representations at any instant of decision making.

Postulate 5 (Non-Determinism): Several feasible designs can be generated to the level specified by the designer.

1.6.4 COMPUTATIONAL ANALYSIS OF DESIGN (CHAPTERS 5-7)

One common issue for all designs is the complexity of the design process. The complexity of the design process stems from the nature, variety, and mutual interdependence of the choices available to the designer in the course of the design process. Consequently, the issue of complexity is studied in FDT from three perspectives that represent a spectrum of problems, which are related to the architecture of design systems:

- The architecture of the *interface* (analysis and synthesis) between the artifact and function spaces (Chapter 5).
- The architecture of a real design process; which is viewed as stepwise, iterative, and evolutionary (Chapter 6).
- The architecture of *basic synthesis* activities, which support the real design process (Chapter 7).

1.6.5 THE MEASUREMENT OF A DESIGN STRUCTURAL AND FUNCTIONAL COMPLEXITY (CHAPTERS 8-9)

Information theory and cognitive psychology are utilized to quantitatively describe a universal set of metrics, which are used for measuring the functional and structural complexity of design artifacts and design processes. We show that assembly time can be predicted from these measures. The functional and structural complexity measures constitute *ad hoc* engineering metrics without a corresponding scientific foundation. Therefore, we contend that by analogy with thermodynamics, scientific design evaluation tools may be developed. By applying large-scale or macroscopic formulas we attempt to understand, or at least quantitatively assess, the microscopic design process.

1.6.6 ALGORITHMIC METHODS AND DESIGN APPLICATIONS (PARTS III & IV)

The FDT theory leads to two useful outcomes. First, heuristic methods (including *group technology*, *case based reasoning*, *genetic algorithms*, *adaptive learning for successful design*, and *continuation methods for maintaining design consistency*) are described for linking the design process to its underlying knowledge base (Part III). Second, powerful and comprehensive examples (methodological validation) have been developed to show that the proposed theory is consistent and fruitful (Part IV).

Based on the theory (Part II), we describe in chapter 10 a computer CADAT (Case-based Design Advisory Tool) methodology. CADAT is a case-based reasoning system that assists in determining efficient appropriate groupings and configurations of parts in order to fulfill customer requirements. The CADAT methodology is implemented by Case-1™. Case-1 is a powerful knowledge-based problem resolution tool used to assist in solving issues typically encountered in product support or help desk roles. Case-1 uses case-based reasoning to best match a list of requirements with a library of existing sub-requirements and/or structural properties.

1.7 CONCLUDING REMARKS

The field of formal design theory presented in this book is still in its early stages of development. As in other mathematical sciences, formal design theory needs to follow an evolutionary development process, and not be concerned with whether results conform with views held for a long time. Thus, what is initially important to us is the gradual development of a design theory that is based on the careful analysis of the ordinary everyday interpretations of design facts. This preliminary stage of transition from nonmathematical plausibility considerations to the formal procedure of mathematics is necessarily heuristic. Its first applications (which serve to

™ Case-1 is a trademark of ASTEA International, 1995.

corroborate the theory) are to elementary design situations where no theory is required. However, an awareness of the following issues may be helpful for future research on the more advanced aspects of the theory [48, 49]:

1. The critical role of needs assessment, which precede active design;
2. The application of prior proven concepts and solutions to innovative design;
3. The robustness of product and manufacturing processes;
4. The design as a social process (involving designers, customers, and others) that consists of creating and refining a "shared meaning" of requirements and potential solutions through continual negotiations.

The approach taken in this book can provide a common framework for various engineering design problems, and thus improve the semantic pollution prevailing in engineering design. As engineering science breaks into subgroups and less communication is possible among the disciplines; the growth of total knowledge is likely to diminish. It is one of the main objectives of mathematical design theory to develop a framework that enables different specialists to better interpret the design communications of others. Thus, the engineers that realize the strong formal similarity between the design process and either topology in mathematics or problem solving in cognitive psychology may be in a better position to learn from the mathematicians or psychologists than those who do not.

The next stage is to apply the theory to somewhat more complicated, less obvious situations. Beyond this lies real success: genuine prediction by theory, and fully reliable applications in the form of intelligent CAD tools for the design engineer. It is well known that all mathematized sciences have gone through these successive phases of evolution.

REFERENCES

1. Antonsson, E.K., "Development and Testing of Hypotheses in Engineering Design Research," *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 109, pp. 153-154, 1987.
2. Bezier, P.E., "CAD/CAM: Past, Requirements, Trends," In *Proc. CAD*, Brighton, pp. 1-11, 1984.
3. Braha, D. and Maimon, O., "A Mathematical Theory of Design: Modeling the Design Process (Part II)," *International Journal of General Systems*, Vol. 26 (4), 1997.
4. Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M. and Gero, J.S., *Knowledge-Based Design Systems*. Reading, MA: Addison-Wesley, 1990.
5. Cross, N. (ed.), *Development in Design Methodology*. New York: John Wiley, 1984.
6. Dasgupta, S., "The Structure of Design Processes," In *Advances in Computers*, Vol. 28, M.C. Yovits (ed.). New York: Academic Press, pp. 1-67, 1989.
7. Dixon, J.R., *AI EDAM*, Vol. 1 (3), pp. 145-157, 1987.
8. Freeman, P. and Newell, A., "A Model for Functional Reasoning in Design," In *Proc. of the 2nd Int. Joint Conf. on Artificial Intelligence*, pp. 621-633, 1971.
9. Gero, J.S., "Prototypes: A New Schema for Knowledge Based Design," *Technical Report*, Architectural Computing Unit, Department of Architectural Science, 1987.
10. Giloi, W.K. and Shriver, B.D. (eds.), *Methodologies for Computer Systems Design*. Amsterdam: North-Holland, 1985.
11. Glegg, G.L., *The Science of Design*. Cambridge, England: Cambridge University Press, 1973.
12. Hubka, V., *Principles of Engineering Design*. London: Butterworth Scientific, 1982.
13. Hubka, V. and Eder, W.E., *Theory of Technical Systems: A Total Concept Theory For Engineering*

Design. Berlin: Springer-Verlag, 1988.

14. Ishida, T., Minowa, H. and Nakajima, N., "Detection of Unanticipated Functions of Machines," In *Proc. of the Int. Symp. of Design and Synthesis*, Tokyo, pp. 21-26, 1987.
15. Jaques, R. and Powell, J.A. (eds.), *Design: Science: Method*. Guildford, England: Westbury House, 1980.
16. Jones, J.C., *Design Methods: Seeds of Human Futures* (2nd Edition). New York: John Wiley, 1980.
17. Kuhn, T.S., Postscript - 1969. In *The Structure of Scientific Revolutions*. Chicago, IL: University of Chicago Press. Enlarged 2nd Edition, pp. 174-210, 1970.
18. Maher, M.L., "A Knowledge-Based Approach to Preliminary Design Synthesis," *Report EDRC-12-14-87*, Carnegie Mellon University Engineering Design Research Center, 1987.
19. Maimon, O. and Braha, D., "A Mathematical Theory of Design: Representation of Design Knowledge (Part I)," *International Journal of General Systems*, Vol. 26 (4), 1997.
20. Maimon O. and D. Braha, "On the Complexity of the Design Synthesis Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 26 (1), 1996
21. Murthy, S.S. and Addanki, A. "PROMPT: An Innovative Design Tool," In *Proc. of the 6th Nat. Conf. on Artificial Intelligence*, Seattle, WA, 1987.
22. Paynter, H.M., *Analysis and Design of Engineering Systems*. Cambridge, MA: MIT Press, 1961.
23. Penberthy, J.S., *Incremental Analysis and the Graph of Models: A First Step Towards Analysis in the Plumber's World*, S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, 1987.
24. Ressler, A.L., "A Circuit Grammar for Operational Amplifier Design," *Technical Report 807MIT*, Artificial Intelligence Laboratory, 1984.
25. Rieger, C. and Grinberg, M., "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms," In *Proc. of the 5th Int. Joint Conf. on Artificial Intelligence*, pp. 250, 1977.
26. Rinderle, J.R., "Function and Form Relationships: A basis for Preliminary Design," *Report EDRC-24-05-87*, Carnegie Mellon University Engineering Design Research Center, Pittsburgh, PA, 1987.
27. Roylance, G., "A simple Model of Circuit Design," *Technical Report 703*, MIT Artificial Intelligence Laboratory, 1983.
28. Rychener, M. (ed.), *Expert Systems for engineering design*. New York: Academic Press, 1988.
29. Shina G.S., *Concurrent Engineering and Design for Manufacture of Electronics Products*. Van Nostrand Reinhold, 1991.
30. Simon, H.A., *The Science of the Artificial*. Cambridge, MA: MIT Press, 1981.
31. Spillers, W.R. (ed.), *Basic Questions of Design Theory*. Amsterdam: North-Holland, 1972.
32. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.
33. Tong, C. and Sriram, D. (eds.), *Artificial Intelligence Approaches to Engineering Design*, 1991.
34. Ulrich, K.T., "Computation and Pre-Parametric Design," *Technical Report 1043*, MIT Artificial Intelligence Laboratory, 1988.
35. Winston, P.H., et. al., "Learning Physical Descriptions From Functional Definitions, Examples and Precedents," *Memo 679*, MIT, Artificial Intelligence Laboratory, 1983.
36. Arciszewski, T., "Design theory and methodology in Eastern Europe," In *Design Theory and Methodology-DTM'90* (Chicago, Il), pp. 209-218, New-York, NY, The American Society of Mechanical Engineers, 1990.
37. Braha, D. and Maimon, O. "The Measurement of A Design Structural and Functional Complexity," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 28 (3), 1998.
38. Dijkstra, E.W. Notes on Structural Programming. in O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, *Structural Programming*. Academic Press, New York. 1972.
39. Eder, W.E., "Engineering Design - a perspective on U.K. and Swiss development," In *Design Theory and Methodology-DTM'90* (Chicago, Il), pages 225-234, New-York, NY, The American Society of Mechanical Engineers, 1990.
40. Finger, S. and Dixon, J. R., "A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes," *Research in Engineering Design*, Vol. 1(1), pp. 51-67, 1989.
41. Hundal, M.S., "Research in design theory and methodology in West Germany," In *Design Theory and Methodology-DTM'90* (Chicago, Il), pages 235-238, New-York, NY, The American Society of Mechanical Engineers, 1990.

42. Klir, J.G. *Architecture of Systems Problem Solving*. Plenum Press. New York. 1985.
43. Maimon, O. and Braha, D. "A Proof of the Complexity of Design," *Kybernetes: An International Journal of Cybernetics and General Systems*, Vol. 21 (7), pp. 59-63, 1992.
44. Maimon, O. and Braha, D., "An Exploration of the Design Process," *Technical Report*, Boston University, 1994.
45. Suh, N. P. "Development of the science base for the manufacturing field through the axiomatic approach." *Robotics & Computer-Integrated Manufacturing*, Vol. 1(3/4), pp. 397-415, 1984.
46. Tomiyama, T., "Engineering design research in Japan," In *Design Theory and Methodology-DTM'90* (Chicago, IL), pages 219-224, New-York, NY, The American Society of Mechanical Engineers, 1990.
47. Warfield, J.N. *A Science of Generic Design*. Intersystems Publications, Salinas, CA. 1990.
48. Barkan, P. and Hinckley, C. M., "Limitations and Benefits of Structured Methodologies," *Manufacturing Review*, Vol. 6 (3), 1993.
49. Reich, Y., "The Development of Bridger: A Methodological Study of Research on the Use of Machine Learning in Design," *Artificial Intelligence in Engineering*, Vol. 8, 1993.
50. Nadler, G. *The Planning and Design Approach*. John Wiley. New York.
51. Braha D. and Maimon O., "The Design Process: Properties, Paradigms and Structure" *IEEE Transactions on Systems, Man and Cybernetic (Part A)*, Vol. 27 (3), 1997.

CHAPTER 2

DESIGN AS SCIENTIFIC PROBLEM-SOLVING

Following Proclus' aphorism that "it is necessary to know beforehand what is sought," a ground rule of intellectual endeavor seems to be that any new field of study, to be recognized properly, must first scrutinize its bounds and objectives: where it stands in the universe and how it proposes to relate to the established disciplines. Such clarification is the object of this chapter.

In this chapter, we examine the logic and methodology of engineering design from the perspective of the philosophy of science. The fundamental characteristics of design problems and design processes are discussed and analyzed. These characteristics establish the framework within which different design paradigms are examined. Following the discussions on descriptive properties of design, and the prescriptive role of design paradigms, we advocate the plausible hypothesis that there is a direct resemblance between the structure of design processes and the problem solving of scientific communities. The scientific community metaphor has been useful in guiding the development of general purpose, highly effective, design process meta-tools [73].

2.1 INTRODUCTION

2.1.1 MOTIVATION AND OBJECTIVES

Design as problem solving is a natural and most ubiquitous of human activities. Design begins with the acknowledgment of needs and dissatisfaction with the current state of affairs and realization that some action must take place in order to solve the problem, so scientists have been designing and acting as designers (sometimes unconsciously) throughout their lives. As such, it is of central concern to all disciplines within the artificial sciences (engineering in the broad sense).

Design science is a collection of many different logically connected knowledge and disciplines. Although there is no single model that can furnish a perfect definition of the design process, design models provide us with the powerful tools to explain and understand the design process. Design has been discussed, among others, in contexts such as general design methodologies [105, 52, 108, 36, 6, 21, 22], design artifacts representation [30, 48, 94, 122, 92, 83], computational models for the design

process [78, 84, 91, 96, 120, 71], knowledge-based CAD systems [32, 117, 97] and design theories [46, 112, 124, 72, 73, 13].

Our research in engineering design [13, 72, 73] has led us to believe that *evolution* is fundamental to design processes and their implementation by computer-aided design (CAD) and “expert” design systems in many domains. In spite of the disparity between the models, and regardless of whether one is designing computer software, bridges, manufacturing systems or mechanical fasteners, evolutionary speaking they are similar. As the design process develops, the designer modifies (due to bounded rationality) either the tentative (current) design, or the specifications - based on new information obtained in the current design cycle. The modification is performed in order to remove discrepancies, and eventually establish a fit between the two parts. The evolved information reflects the fundamental feature of bounded rationality. The new information determines the tentative design knowledge, stating the relation among high and low levels of design specifications. It also determines the inference rules (or inference mechanism) that specify the method for deriving new design specifications and/or design artifacts. Both the sets of design knowledge and inference rules reflect the beliefs, skills and expertise unconsciously developed by designers through the repetitive experiences. The converging design process includes a testing stage for verifying the tentative design against the tentative specifications to establish the direction of their future elaboration. This process terminates with an acceptable design. These characteristics were arrived at from arguments based on the concept of “bounded rationality” [106].

In this chapter, we present a largely philosophical discussion of our motivations. We focus our attention on how *scientific communities* solve problems. Our thesis is that design as an evolutionary problem solving activity conforms to the structure of problem solving of scientific communities. That scientific communities are successful at generating and deciding between alternative explanations for phenomena is indisputable. Scientific progress, looked at globally and with a time scale of many decades seems coherent and purposeful. At any one time many conflicting theories and paradigms may support to explain the same phenomenon. Scientific communities themselves can be the subject matter of scientific research. The nature of science has been a fertile topic in philosophy from the pre-Socratic through the present day. We are particularly indebted to a number of philosophers and historians of science of this century among them Popper’s, Kuhn’s, Laudan’s and Lakatos [86, 57-60, 66, 61-63]. We hope to gain insight from this research that will be useful in guiding the development of general purpose, highly effective design process meta-tools [73].

2.1.2 OVERVIEW OF THE CHAPTER

Section 2.2 scrutinizes the bounds and objectives of design from the perspective of the design problem. The basic characteristics as articulated in this section are:

1. Generally, designers act and behave under conditions of bounded-rationality;
2. Alternatives, options and outcomes are usually not given in advance (ill-

- structured problems), and must be found and developed by some research process;
3. Usually, the optimum decisions will not be sought and satisfying decisions will fully be accepted;
 4. Computationally speaking, most design optimization problems (well-structured problems) are intractable. Hence, the optimal decisions will generally not be sought and satisfying decisions will fully be accepted.

As a result of these basic postulates, we argue in Section 2.3 that the design process can be viewed as a stepwise, iterative, evolutionary transformation process. These characteristics establish the framework within which different design paradigms are examined. Section 2.4 glean useful ideas from the metaphor of scientific research to define design paradigms. Having defined a design paradigm, we survey the contemporary design paradigms. All the paradigms share the characteristic of observed evolutionary phenomenon which occurs between the time when a problem is assigned to the designer and the time the design is passed on to the manufacturer.

Following our previous discussions on descriptive properties of design, especially the adaptive and evolutionary properties discussed in Section 2.3, and the prescriptive role of design paradigms (Section 2.4), we pose in Section 2.5 the hypothesis that there is a direct and striking resemblance between the structure of design processes and the structure of problem solving of scientific communities. The basic correspondence is summarized as follows:

1. The counterpart of the Kuhnian paradigm or Laudan's research tradition is the designer's knowledge-base needed to generate the set of design solutions;
2. The counterpart of a set of phenomena, events or problems are design problems that are entirely characterized by and generated as a result of measurable and non-measurable requirements (specifications);
3. The counterpart of a scientific theory (set of hypotheses) is the tentative design/form serving (much the same as scientific theories) as a vehicle for the designer to capture her thoughts;
4. Scientific discovery follows the hypothetico-deductive method, or the more justifiable procedure (following Popper) of conjecture and refutation. It is with a direct correspondence with the evolutionary nature of design processes;
5. Incremental redesign activity corresponds to the continual and incremental evolution of scientific theories within a normal science, whereas innovative redesign activity corresponds to a transition to a new paradigm (conceptual or paradigm-shift).

Regardless of whether or not the scientific community metaphor serves as the bases for explanations for the evolutionary design process, it has also a heuristic value in explicitly carrying out the act of design. In Chapter 6, we develop a model of the process based on double interleaved activities of *analysis* and *synthesis*, which explode the specification world (the counterpart of scientific phenomena), and the

design artifact (the counterpart of a scientific theory), until a successful solution is achieved. We illustrate the application of this evolutionary design model, among others, to the design of mechanical fasteners (Chapter 6), and gearbox (Chapter 17). Section 2.6 outlines a design methodology, based on the scientific community metaphor, by emphasizing the variational (or parametric) design part. Section 2.7 concludes the chapter.

2.2 PROPERTIES OF THE DESIGN PROBLEM

2.2.1 THE UBIQUITY OF DESIGN

The natural point to begin any discussion of design is to state succinctly in a single sentence what it is that one does when one designs and what the end product is. Such an endeavor has been attempted in a variety of contexts including architecture, engineering and computer science. Clearly, an over-simplified or single sentence definition of design will not do. One reason why definitions fail is the omnipresence of design or problem solving as a natural human activity [146]. We have been designing and acting as designers (sometimes unconsciously) throughout our lives. Designing is pervasive in many human activities, for example an engineer conceiving of a new type of toaster or configuring a manufacturing cell, a financial manager configuring a profitable portfolio, or a cook concocting a new pizza. Underlying these design tasks is a core set of principles, rules, laws and techniques that the designer uses for problem solving. According to common sense, design is the process of putting together or relating ideas and/or objects in order to create a whole which hopefully achieves a certain purpose [19]. Design, according to the Encyclopedia Britanica, “is a process of developing plans as schemes of actions; more particularly a design may be the developed plan or scheme, whether kept in mind or set forth as a drawing or model... Design in the fine arts is often considered to be the creative process per se, while in engineering, on the contrary, it may mean a concise record of embodiment of appropriate concepts and experiences. In architecture and product design the artistic and engineering aspects of design tend to merge; that is; an architect, craftsman, or graphic or industrial designer cannot design according to formulas alone, nor as a freely as can a painter, poet, or musician.” In its effort to promote research in the field, the National Science Foundation defines design as “the process by which products, processes, and systems are created to perform desired functions, through specifications.” These specifications include desired object features, functions, constraints, etc. Another broad definition is that design is any arrangement of the world that achieves a desired result for known reasons. The process of design itself involves some of the same constraints as diagnostic processes or planning processes. Design approaches have traditionally been subjective; that is, a standardized set of rules is not readily available which can be applied to all classes of design problems.

2.2.2 DESIGN AS A PURPOSEFUL ACTIVITY

Design begins with the acknowledgment of needs and dissatisfaction with the current state of affairs and realization that some action must take place in order to correct the problem. Most design theorists, including [105, 4, 67, 98], have derived a number of consequences of this ostensibly intuitive observation:

- There is a distinction between engineering science (the 'science of the artificial' as Simon coined) and natural science (e.g. physics, chemistry and biology) that can be expressed in a variety of ways. First, the aims and methodology of natural science and engineering differ. That is, natural science is concerned with 'analysis' and engineering with 'synthesis' [22]. Second, natural science is 'theory-oriented' while engineering is 'result-oriented'; and third, the engineering activity is creative, spontaneous and intuitive, while science is rational [146, 98];
- Design is a pragmatic discipline concerned with how things should be done. Thus, the design activity is influenced by the designer's world view and values. Consequently, the recognition and identification of the design problem, the nature of the design solution and the determination of valid research topics in engineering design, are all intimately a function of the designer's perspective [22].

2.2.3 DESIGN IS A TRANSFORMATION BETWEEN DESCRIPTIONS

Louis Kahn, the famous architect, viewed design as a process by which the transcendent forms of thinking and feeling produce the realization of form. By form, Kahn meant the essence created by a certain relationship of elements within the whole. Thus, in practical terms, a design problem is characterized in terms of a set of requirements (specifications, goals and constraints) such that if an artifact or system satisfies the requirements and is implemented according to the proposed design, the design problem will be solved [93, 76, 111].

2.2.4 CATEGORIES OF DESIGN REQUIREMENTS

The most basic type of requirement is empirical, measurable or well-defined in nature. A requirement is well-defined when it specifies externally observable or empirically determinable qualities for an artifact [22]. Some requirements can naturally be stated as empirical, which means that one knows precisely what procedures to construct or use in order to determine whether or not a given design meets such requirements. Design problems that are entirely characterized by such requirements fall within the category of what Simon [102] termed well-structured problems. The most important varieties of well-defined requirements are functionality, performance, reliability and modifiability [146]. Functional

requirements refers to the capability of the designed artifact to do certain desirable things [22], that is, the minimum set of independent specifications that completely define the problem. Thus, the functional requirements are the non-negotiable characteristics of the desired solution. We distinguish between functionality and behavior as different levels of description, where the function of a piece of a system relates the behavior of that piece to the function of the system as a whole. Performance refers to the competence of the desired artifact to achieve its functionality well. In practical terms, it usually refers to economy in the use of some observable set of resources. Reliability of artifacts is defined as the probability that the artifacts will conform to their expected behavior throughout a given period of time [146]. Modifiability refers to the ease with which changes may be incorporated in the design of artifacts [22]. Modifiability requirements completely support the evolutionary characteristic of the design process, and the act of successive changes or improvements to previously implemented designs.

A design problem may also be generated as a result of requirements that are not measurable. Such requirements are termed as ill-defined requirements (conceptual), and any reasonably interesting and complex design problem will contain ill-defined requirements. A design problem produced fundamentally as a consequence of a set of ill-defined requirements is referred to as an ill-structured design problem [102]. The initial requirements may be neither precise nor complete. Hence, in order to show that a design solution satisfies a set of initial requirements, (including ill-defined objectives), all requirements must eventually be converted into well-defined requirements. The process by which this information is transformed into well-defined design objectives is called the design requirements extraction process. Hence, the extraction, elaboration or refinement of requirements is an inherent and integral part of the generation of design [22].

2.2.5 BOUNDED RATIONALITY AND IMPRECISENESS OF DESIGN PROBLEMS

Decision making during the design activity deals with highly complex situations. The traditional methods of decision-making are based on the classical model of pure rationality, which assumes full and exact knowledge about the decision situation being considered. In design, assumptions about the exact knowledge are almost never true. At least to a large measure, the requirements are not comparable and therefore, the preference ordering among them is incomplete. The departure from 'pure-rationality' based methods is needed in design because of the fact that the designer has a limited information-processing capacity and the information is vague. Generally, designers act and behave under conditions of 'bounded-rationality' [104, 106]. The concept of bounded rationality was developed by Simon in the context of administrative decision making [104], and subsequently elaborated inter alia to design decision-making. Such limitations may arise in several ways: the designer may not know all the alternative sequence of decisions; or even assuming all the conditions are known, the designer may be unable to decide the best sequence of

decisions ; or finally, the time and cost of computing the best possible choices may be beyond the bounds of the available resources.

2.2.6 THE SATISFICING NATURE OF DESIGN PROBLEMS

The bounded rationality led Simon to postulate that, more often than not, the optimum decisions will not be sought and satisfying decisions will fully be accepted. That is, instead of requiring an optimal design, designers accept a “good” or “satisfactory” one. In Simon’s terms this attitude toward design, which allows the use of heuristic methods, is called ‘satisficing’. The postulate of satisfying decisions is related to the psychological theory of ‘aspiration level’ given in the classical work of [68]. Another related concept is ‘incrementalism’ given by [69]. ‘Incrementalism’ is also based on the limited information-processing capacity of the decision-makers (designers) which forces them to make decisions similar to those previously made.

2.2.7 THE INTRACTABILITY OF DESIGN PROBLEMS

Optimization theory is applied as a recognized technique that can assist designers in the decision-making process of design. Utilizing optimization theory to solve design problems poses optimization problems which demonstrate inherent intractability. Typical instances of design optimization problems include:

1. Design of mechanisms employs graph enumeration and graph isomorphism problems are known to be NP-complete;
2. Design of printed circuit boards (PCB) includes partitioning, placement and routing problems, which are known to be intractable. Such problems are referred to as NP-complete, or Non-deterministic Polynomial time Complete problems [31]. The CPU time required to solve an NP-complete problem, based on known algorithms, grows exponentially with the “size” of the problem. There exist no polynomial time transformations for NPC problems nor are there any polynomial time algorithms capable of solving any NP problems, therefore these problems are considered to be “open” or unsolved problems. The potential to solve these NP and NPC problems depends on the availability of certain heuristics. Hence, in spite of knowing that there does indeed exist an optimal solution to a design problem, the designer may still resort to satisficing methods.

2.2.8 THE FORM OF DESIGN

Designing an artifact can be considered a transition from concepts and ideas to concrete descriptions. By form (a synonym to design) we mean the essence or ultimate output of a design process created by a certain relationship of elements within a whole. For example, the form of a piston for a model aircraft engine, is a

piece of short cylinder designed to fit closely and move inside another cylinder or tube. The piston consists of a cylinder, piston rod and pin. Despite whether it is made of plastic, iron or steel, it is recognized as a piston as long as the cylinder, piston, and pin remain in a certain relationship to one another.

The concept of form is elusive, abstract and complex. The design process involves conceiving of the concepts relevant to the form and the relationships between them, and representing the concepts using specific well-defined language. In the case of engineering design, such design descriptions range from specifications in formal language (such as computer-aided engineering systems, symbolic programming techniques associated with AI and hardware design/description languages) through description in quasi-formal notation (such as linguistic descriptions and qualitative influence graphs) to very informal and visual descriptions (such as functional block diagramming, flow-Diagrams and engineering drawings). The concepts underlying a design are captured in three views: The functional view describes the design's functions and processes, thus connecting its capabilities. This view also includes the inputs and outputs of the activities, i.e., the flow of information to and from the external activities. For example, in the design process of integrated circuits the functional level includes a register-transfer diagram. The behavioral view describes the design's behavior over time, the states and modes of the design, and the conditions and events that cause modes to change. It also deals with concurrency, synchronization and causality. Good examples are constraints that components must satisfy such as timing properties. The behavioral and functional views are invariant characteristics of the design or form. The structural view describes the subsystems and modules constituting the real system and the communication between them. It also captures geometrical information. While the two former views provide the conceptual model of the design, the structural view is considered to be a physical model, since it is concerned with the various aspects of the system's implementation. As a consequence, the conceptual model usually involves terms and notions borrowed from the problem domain, whereas the physical model draws more upon the solution domain. Examples include details about materials, layout, process parameters, heat conductivity and other physical parameters.

The design/form serves several distinct roles in the development of an artifact. First, a design/form constitutes a tangible representation of the artifact's conceptual and physical properties, and thus serves as a vehicle for the designer to visualize and organize thoughts. Second, it serves as a plan for implementation. To accomplish this, the design/form should contain a systematic representation of the functional relationships of the components. Such demarcation of form/design and implementation has not always been necessary [146]. Jones [52] and Ferguson [147, pp. 3-4] have mentioned that the artisans of the 18th and 19th century did not demarcate between conceptualizing an artifact and making it; and that the transition from "designing without drawings" to the engineer's way of "designing with drawings" is ascribed mainly to the increasing complexity of modern devices (such as an internal-combustion engine), and the need to enhance the interaction between the client who wanted a machine built and those who would build the machine [147].

Third, the design description must also serve as a document (for instance, in the form of user-manuals) that describe how to harness the final artifact by the user. Finally, the form/design serves as a vehicle for reflecting the evolutionary history that led to the emergence of the final form/design, thus facilitating the inspection, analysis and redesign (change) of the artifact [22].

2.3 PROPERTIES OF THE DESIGN PROCESS

2.3.1 SEQUENTIAL AND ITERATIVE NATURES OF DESIGN

Many design theorists argue that the design process can be viewed as a stepwise, iterative, evolutionary transformation process [105, 124, 112]. Consider the following two assertions (with justifications) regarding the nature of the typical design process:

Assertion #1: Design is a sequential process

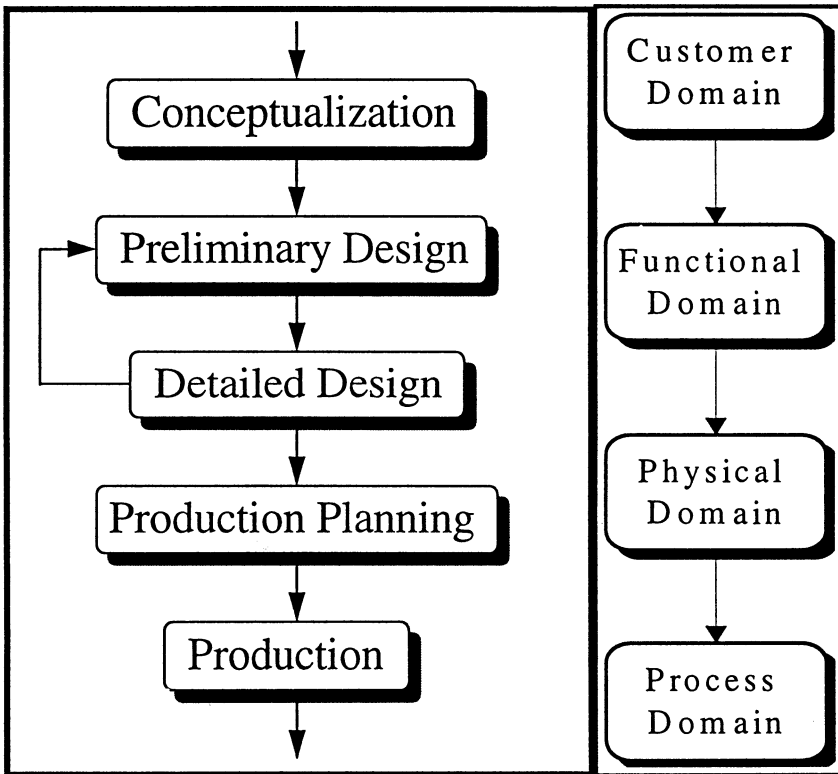
Almost every flowchart ever created that has attempted to describe the design process has shown evidence of the fact that design is a sequential process (see Figure 2.1). The design process evolves from concept through realization and it is impossible to go backwards. A part cannot be assembled until the components are machined; the components cannot be machined until the NC code is created; the NC code cannot be created without a dimensioned part model; the part model cannot be dimensioned without a set of requirements and a general notion of what the part looks like; and presumably the last two items come from a need that must first be identified. All this points to the seemingly undeniable truth that there is an inherent, sequential order to most design processes.

Assertion #2: Design is an iterative process

One can reason equally effectively, however, that design is an iterative process. First, designers are only human and have a bounded rationality. They cannot simultaneously consider every relevant aspect of any given design. As the design process progresses, new information, ideas, and technologies become available that require modifying the design. Second, design systems are limited; there is no known system that can directly input a set of requirements and yield the optimum design. Rather, the designer must iteratively break down the set of requirements into dimensions, constraints, and features and then test the resulting design to see if the remaining requirements were satisfied (see Figure 2.2). Finally, the real world often responds differently than is imagined. The real world is full of chaotic reactions that are only superficially modeled in any design system. All this points to the seemingly undeniable truth that there is an inherent, iterative nature to the design process.

In order to reconcile these two disparate visions of the design process, we categorize design iteration as occurring either *between* design stages (*inter-stage*

iteration) or *within* a design stage (*intra-stage* iteration) and then create a new model of the design process combining both approaches to design (Figure 2.3). In this model, design still flows sequentially from initial concept through realization, each design stage providing the data and requirements for the subsequent stage. Within each design stage, however, the designer iteratively creates a design that meets the given requirements. This model largely represents the current state-of-the-art in CAD/CAM/CAE systems. While there are numerous software modules to assist the designer during intra-stage design iteration (e.g., QFD software to help identify customer needs and CAE software to analyze a current design), the tools are generally not well integrated at the inter-stage level.



A. [145]

B. [144]

Figure 2.1 Traditional Views of Mechanical Design

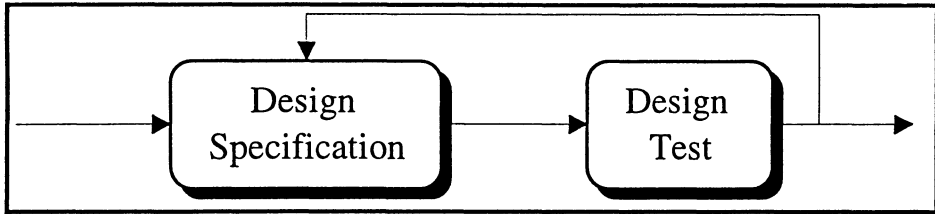


Figure 2.2 Specification and Test Iteration

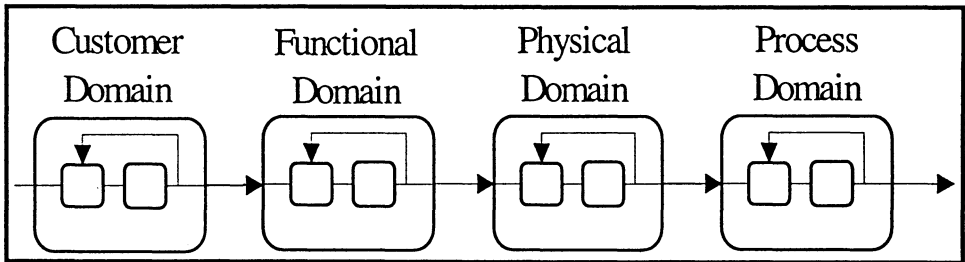


Figure 2.3 Combining Sequential and Iterative Design

2.3.2 THE EVOLUTIONARY NATURE OF THE DESIGN PROCESS

The concepts underlying the evolutionary characteristic of design are captured in three views: purposeful adaptation of artificial things, *ontogenetic*¹ design evolution and *phylogenetic*² design evolution (both latter phrases are borrowed from biology; [38] and [148]). Purposeful adaptation, according to Simon, can be thought of as an interface between the “inner” environment, the substance and organization of the artifact itself, and an “outer” environment, the surroundings in which it operates. If the inner environment is appropriate for the outer environment, or vice versa, the artifact will serve its intended purpose. For instance, a ship’s chronometer reacts to the pitching of the ship only in the negative sense of maintaining an invariant relation of the hands on its dial to the real time, independently of the ship’s motions. Regardless of whether or not the adaptation model is a universal feature of artificial systems, it also has a heuristic value. Hence, we can often predict behavior from knowledge of the artifact’s goals and its outer environment with only minimal assumptions about the inner environment.

Ontogenetic design evolution refers to the design processes that share the

¹ Ontogeny: The life history of an embryonic individual

² Phylogeny: The evolutionary history of a lineage

characteristic of observed evolutionary phenomenon which occurs between the time when a problem is assigned to the designer and the time the design is passed on to the manufacturer [22]. During this period the design evolves and changes from the initial form to the acceptable form. In this case, we say that there is a fit between the design and the requirements. The evolutionary model of design seems to support the cognitive model of design: Yoshikawa [124] argues that the ontogenetic design process can be decomposed into small design cycles. Each cycle has the following sub-processes:

1. Awareness - problem identification by comparing the object under consideration and the specifications;
2. Suggestion - suggesting the key concepts needed to solve the problem;
3. Development - developing alternatives from the key concepts by using design knowledge;
4. Testing - evaluating the alternatives in various ways such as structural computation, simulation of behavior, etc. If a problem is found as a result of testing, it also becomes a new problem to be solved in another design cycle;
5. Adaptation - selecting a candidate for adaptation and modification.

Protocol studies on how technically qualified people design were conducted by several researchers [e.g., 2, 37, 119, 53]. Subjects were given problems to solve in a specified amount of time and told to talk aloud while they were developing the design. Based on these studies, the researchers formulated several models of the design process. However, in spite of the disparity between the models, evolutionary speaking they are similar: as the design process develops, the designer modifies either the tentative design or requirements, based on new evidence (information) obtained in the current design cycle, so as to remove the discrepancy between them and establish a fit between the two parts.

Regardless of whether or not the evolutionary model is a universal feature of design processes, the adaptive model has also a heuristic value and serves a useful purpose in explicitly carrying out the act of design. Solving a problem by beginning with a set of goals, identifying subgoals which when achieved realize the goals, then further identifying sub-subgoals that entail the subgoals, and so on, goes by several names in the computer science, cognitive science and AI literature. Goal directed problem solving, stepwise refinement, and backward chaining are notable jargons used [3, 18, 79, 50]. One of the most celebrated of these 'weak' methods is means-ends analysis. This method was proposed by Newell, Simon and associates in the late 1950s and first used in the General Problem Solver (GPS) one of the earliest and most influential systems developed within the problem space/heuristic search paradigm. Means-ends analysis relies on the idea that in a particular task domain, differences between possible initial or 'current' and goal states can be identified and classified. Thus, for each type of difference, operators can be defined that can reduce the difference. Associated with each operator is also a precondition that the current state must satisfy in order for the operator to be applied. Means-ends analysis then attempts to reduce the difference between the current and goal states by applying the

relevant operator. If, however, the preconditions for the operator are not satisfied, means-ends analysis is applied recursively to reduce the difference between the current state and the precondition.

Phylogenetic design evolution refers to the act of redesign, which is defined as the act of successive improvements or changes made to a previously implemented design. An existing design is modified to meet the required changes in the original requirements. A conventional instance of redesign is encountered in discussions of the history of electronic computers where it is convenient to refer to architectural families/computer generations. The members of the family/generation are related to one another through an ancestor/descendant relationship [8, 146]. In general, the concept of computer family/generation is tied directly to advances in technology. For example, vacuum tubes and germanium diodes characterize the first generation, discrete transistors the second and so forth.

The act of redesign can be illuminated and explained by considering two modes of evolution, namely incremental and innovative. The redesign activity may be defined as incremental if

1. Over a long period of time the overall artifact's concept has remained virtually constant;
2. Artifact improvements have occurred through incremental design at the subsystem and component levels and not at the overall system level. That is, there has been no major conceptual shift.

The automobile is an example of an incremental redesign related to an overall artifact's concept. The design team concerned with the next new car will take it for granted that there will be a wheel approximately at each corner and that, more or less, it will have the basic attributes of the Model 'T' [87]. Many other artifacts may be said to fall into the incremental redesign category, for instance, bicycles, tractors, ships and scissors.

Innovative redesign activity is concerned with innovative, novel conceptual design. Pugh and Smith [88] observed that in all probability, while many overall artifact's concepts are fixed, there is a tremendous opportunity for dynamism and innovation at the subsystems and components levels. For example, the differential gear is used in all cars today. There have been tremendous advances in gear technology, manufacturing processes and materials improvements but the concept is static. The innovative redesign activity is followed by incremental redesign activity. Notwithstanding, the limited slip differential is an innovation and improvement of the subsystems level - it is an innovative redesign activity. An innovative redesign is also encountered in the evolution of the ball valve. The first British patent was granted to Edward Chrimes in 1845. This artifact appears to have been conceptually static until the early 1970s with the introduction of the Torbeck valve, and later the Ve Cone valve. As another example, consider the evolution of bicycles which underwent at least seven stages of innovation and improvement of the subsystems level:

1. The pedal system was installed to replace footwork operation, enhance control of

- the wheels, and increase speed;
2. Incremental improvements in technology led to increasing the bicycle's speed;
 3. The increase in speed created difficulties in stopping with feet. Thus, breaks were installed;
 4. Wheel diameter was enlarged to increase speed;
 5. The increase in wheel diameter led to instabilities in the bicycle. Thus, chain transmission systems were installed to increase speed and safety by lowering the need for larger wheel diameters;
 6. Instabilities associated with increased speed and the beating of the wheels against the roads led to the emergence of tires;
 7. In order to enable the rider to have greater control of the pedals, the Free Wheel system was instated which created a more dynamic connection between the pedals and wheels.

There are three additional points to note in this regard. Firstly, the artifacts in the phylogenetic design evolution are mature artifacts that either have been implemented or are operational. Secondly, the time lapse for the entire phylogenetic design evolution is measurable in terms of years (the first ball valve was introduced in 1845, while the first innovative emergence of the Torbeck valve was introduced only in the early 1970s) rather than days, weeks, or months as in the ontogenetic case. Finally, a single cycle of redesign will, in general, by itself constitute one or more cycles of ontogenetic evolution.

2.3.3 DESIGN PROCESS CATEGORIES

Sriram et al. [110] have classified the design process into four categories: creative design, innovative design, redesign and routine design. These classifications of design are process dependent and product independent. In creative design, the domain specific knowledge (e.g. heuristic, qualitative and quantitative) that is needed to generate the solution set and the set of explicit constraints (such as functionality, performance, environmental, manufacturability and resource constraints) may be partially specified, while the set of possible solutions, the set of transformation operators and the artifact space are unknown. Thus, the key element in this design activity is the transformation from the subconscious to the conscious. In innovative design, the decomposition of the problem is known, but the alternatives for each of its subparts do not exist and must be synthesized. Design might be an original or unique combination of existing components. Sriram et al. argue that a certain amount of creativity comes into view in the innovative design process [see also 120]. Redesign is defined as the act of successive changes or improvements to a previously implemented design. An existing design is modified to meet the required changes in the original requirements. In general, two scenarios may lead to the condition of redesign: first, when the design is passed on to the implementer, the artifact may fail to satisfy one or more critical requirements, and thus must be modified so that it satisfies the requirements. Second, the environment for which the artifact had been

originally designed changes (e.g. in technology or other purposes for the artifact differ from those previously assumed) and produces new requirements. In routine design, the artifact's form, its method of design, and its mode of manufacture are known before the design process actually begins. It follows that an a priori plan of the solution exists and that the general nature of the requirements (satisfied by this design) is also a priori known. The task of the designer is essentially to find the appropriate alternatives for each subpart that satisfies the given constraints [110, 14, 15]. Sriram et al. explain that at the creative end of the spectrum, the design process might be spontaneous, fuzzy, chaotic and imaginative. At the routine end of the spectrum, the design is predetermined, precise, crisp, systematic and mathematical.

2.3.4 THE DIAGONALIZED NATURE OF DESIGN

Newer design tools are beginning to affect the stepwise and iterative design process. The technology for both inter-stage and intra-stage categories of iterative design (see Figure 2.3) has become more available. Computer prices are constantly plummeting as their capabilities rise. Design software that used to require an expensive workstation can now run on a personal computer. Design software itself is becoming ever more capable. Higher-end design software packages (e.g., Pro/Engineer or I-DEAS Master Series) allow the designer to create a part and then are able to calculate the NC code to machine it and update the NC code when the part is modified (thereby iterating between the Physical Domain and the Process Domain stages). Recent CAE packages can analyze a part model, calculate key information about the part, and return the designer immediately back to where they were (thereby reducing the intra-stage iteration). Looking ahead, it becomes clear that the model just discussed is exactly backwards from the ideal. Inter-stage iteration is able to respond to conceptual changes and new information and should be fully allowed by the design system. Each inter-stage iteration, however, results in changes that must propagate through the design stages, requiring intra-stage iteration at each stage. As opposed to the aforementioned design model, the ideal design process will, instead, consist of maximizing the inter-stage design iteration and minimizing the intra-stage design iteration.

Maximizing Inter-Stage Design Iteration

In a design model with no inter-stage iteration, design insights are always limited to the current design stage. Because of the inherent iterative nature of design, there has always been a need for design systems that support inter-stage iteration. Recently, however, there have been even more demands made for inter-stage iteration in the form of incremental design.

We are part of a global competitive marketplace that is becoming more global and more competitive every day. Incremental design has become the standard approach towards design in many areas. Most new products are only slightly

modified from their predecessors with slight cosmetic or feature enhancements. In order to decide which new products to develop, large consumer goods companies often create several different prototypes, test market all of them, and develop whichever one sells the best. In this ever-changing environment, fast time-to-market has become critically important. Companies cannot be required to completely redesign a product simply to add new features or modify the specifications or incorporate new materials or new technologies. Computer companies cannot afford to redesign their computer just because a new CPU is introduced. Most of the design specifications do not change. Likewise, in designing a new computer keyboard, many design issues have already been decided including which keys to include and in what order to place them.

Rapid prototyping involves visiting each design stage quickly, in an effort to rapidly create a final product. Changes then are made to the product at each design stage. Rapid prototyping demands productive incremental design. Productive incremental design demands smooth inter-stage iteration. Incremental design begins with a completed design and iterates back to a previous design stage to effect changes on the design. This concept, however, is the antithesis of the sequential design model.

The problem then becomes how to model inter-stage design iteration while acknowledging the sequential nature of design. Towards this goal, we have created the *diagonalized design* paradigm. Diagonalized design reflects the reality that the designer has a bounded rationality and that new information is constantly being gathered during the design process, not simply before each design stage. For example, consider the diagonalized view of mechanical design (see Figure 2.4). Design still progresses from concept through realization, but the designer can incrementally modify the design in any previously defined design stage and the design is automatically updated in all the later design stages.

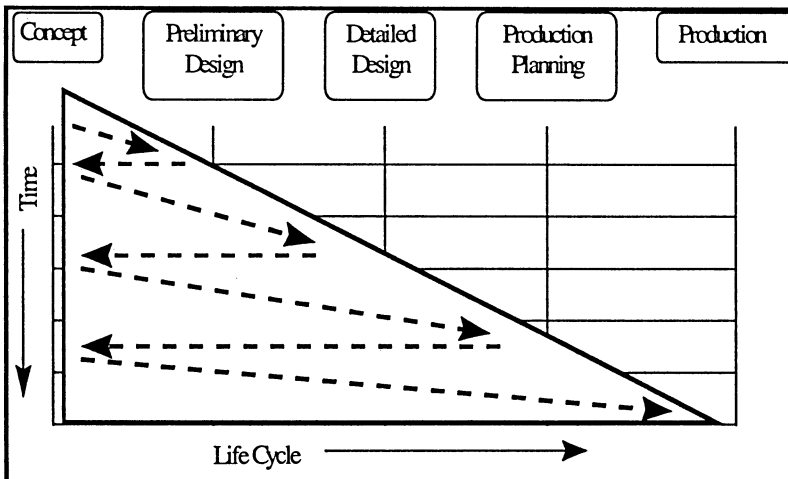


Figure 2.4 Perfectly Flexible Mechanical Design Process

While perfect flexibility in the design process is the goal, the bottom line can be adjusted to reflect more realistic current conditions. The area of the enclosed trapezoid then, becomes a measure of the flexibility of the design process. The flexibility is very dependent upon local conditions. Specific software, available technologies, corporate policies, or other factors greatly affect the flexibility of the design process. These different ranges of flexibility can be shown using diagonalized design. For example, Figure 2.5 represents a limited flexibility, where iteration is restricted to recent design stages; Figure 2.6 shows an inflexible design process where no iteration is allowed; finally, Figure 2.7 demonstrates a design process that is very flexible in the beginning stages of design, but becomes less so as the design moves towards production. By tilting the bottom line the other way, the opposite condition could be shown where beginning stages of design are inflexible, but production is highly flexible.

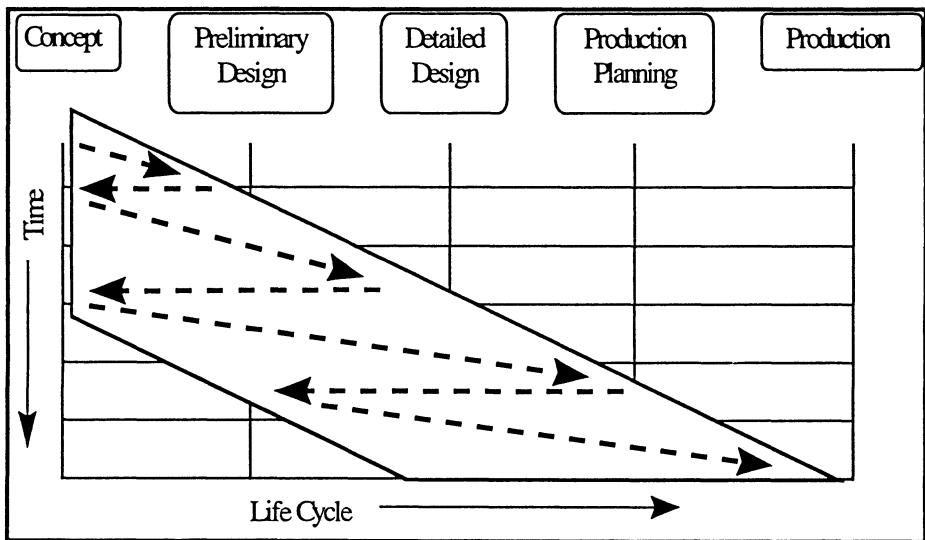


Figure 2.5 Limited Flexibility Mechanical Design Process

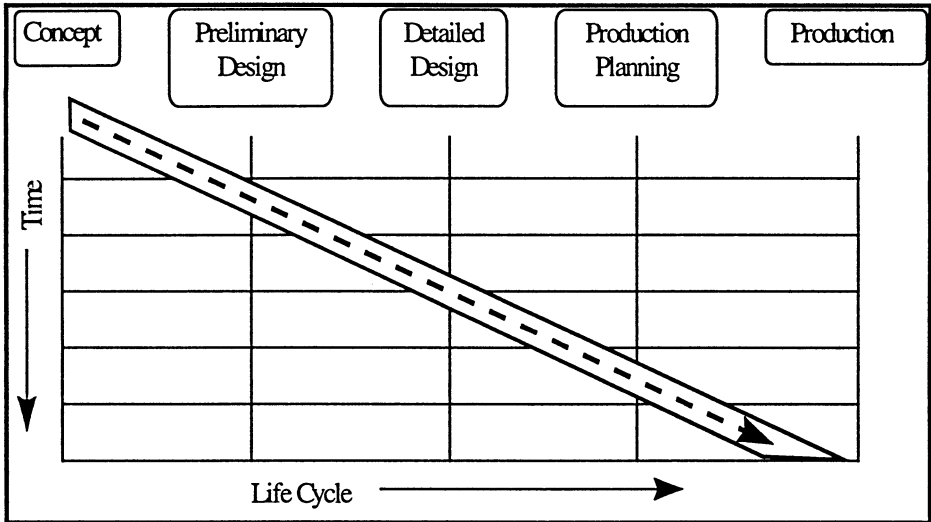


Figure 2.6 Inflexible Mechanical Design Process

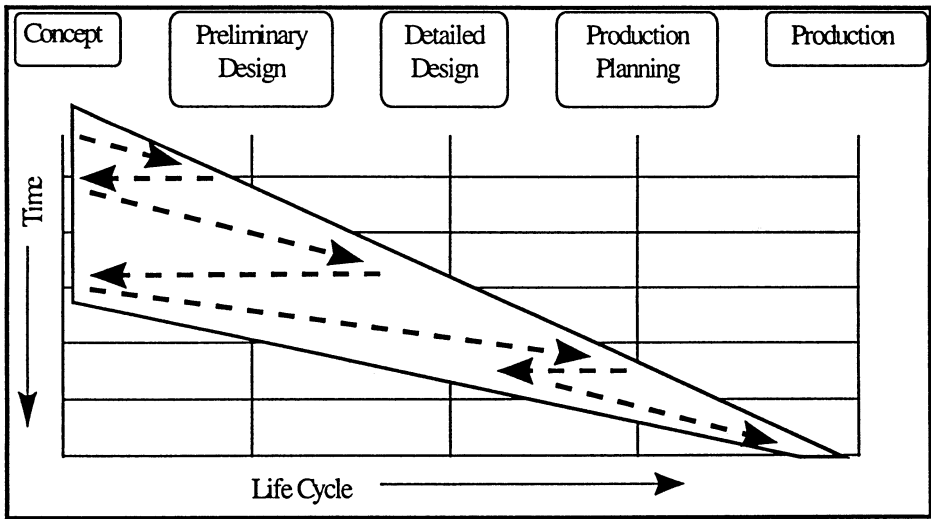


Figure 2.7 Conceptually Biased Flexibility

Design systems capable of fully iterative design will have to support iteration between each set of two sequential design stages:

1. Customer Domain to Function Domain Iteration

2. Function Domain to Product Domain Iteration
3. Product Domain to Process Domain Iteration

The difficulty of implementing iteration in a design system varies both with the level of the iteration as well as the generality of the system.

It is much harder to generalize the design process in earlier iteration levels. Iterating back to the conceptual level of design requires some form of parameterization of the design space. This is certainly possible for very domain-specific design systems but as the generality of the design system increases, however, the general nature of design becomes harder to implement. As a result, there are no known products which adequately address the first level of iteration. There has been much success, however, in generalizing the design process during later stages, both in modeling essentially any type of part and in calculating the NC code to machine arbitrary surfaces. The last level of iteration, however, is highly dependent on the manufacturing process. In one company, this may simply require reprogramming one or more robotic manipulators (the reprogramming could be automated). In this case, smooth iteration would be possible. In another company a product may depend on a highly capital intensive, inflexible design process (e.g., injection molding) and then it becomes harder and more expensive to incorporate design changes. This final stage of iteration may often be difficult to implement, but its implementation is still typically easier to understand than that of earlier iteration levels.

Minimizing Intra-Stage Design Iteration

During intra-stage iteration, the user is simply trying to meet the requirements that were input into the design stage. There are several approaches that could be taken towards minimizing intra-stage design iteration including:

1. Incorporating New Information
2. Modeling Real-World Interactions
3. Allowing Realistic Design Constraints

The three options are addressed through chapters 6, 13, and 14. To illustrate the limitations with how current design systems utilize design constraints, consider the Design-Analysis loop presented in Figure 2.2 as applied to the Detailed Design stage of mechanical design. A part is defined in terms of its dimensions and in a constraint-based design system, the designer enters constraints that the design system satisfies by adjusting the values of the dimensions. During the Design stage, the designer fully defines the dimensions of the design. In the Analysis stage, the designer calculates the values of other desired attributes. Clearly, the only need for the Analysis stage is to calculate whatever attributes cannot be constrained. Furthermore, the fact that the designer analyzes the design indicates that there are degrees of freedom in the design that were artificially constrained in order to analyze the part. Therefore, it can be summarized that a designer is forced to constrain

attributes of the design they do not care about so they can calculate those attributes of the design they do care about. They then incrementally modify the unimportant attributes until the important attributes have achieved their proper values.

2.4 SURVEY OF DESIGN PARADIGMS

2.4.1 DEFINING A DESIGN PARADIGM

According to the dictionary, a paradigm is “a model, a pattern, or a standard.” In [22, 146], it was pointed out that a design discipline may comprise several alternative paradigms at any given time. For instance, when we refer to the process of designing finite-state dynamic systems based on four paradigms: finite-memory machine, Moore machine, Mealy machine and combined machine. All of these paradigms are based on the assumption that one subsystem of the designed structure system is a temporary storage of states of some variables, while the remaining subsystems represent function dependencies among appropriate variables. The paradigms differ in the nature of the function dependencies, which affects the constraints imposed upon the structure of the system to be designed. Another example of the role of paradigms is the notion of functions as building blocks for computer programs which form the basis for the development of a distinct style of programming called functional programming [43].

The common notion of a paradigm was enriched by Thomas Kuhn’s seminal treatise on the nature of the genesis and development of scientific disciplines [57, 58, 59, 60, 75]. The concept of a design paradigm is best elucidated by the Kuhnian paradigm concept as will be illustrated in this section. To Kuhn, a paradigm in its essence comprises of a *Disciplinary Matrix*. A disciplinary matrix refers to a network of theories, techniques, beliefs, values, etc. that are shared by, and generally agreed upon a given scientific community. The following components are identified within a disciplinary matrix [58, 60]:

1. Symbolic Generalizations, examples include Newton’s laws of motions and Ohm’s laws in electricity;
2. Beliefs (or Commitment) in metaphysical and heuristic models, such as the belief that the structure of an atom resembles a tiny planetary system [44], or that logical languages are the most effective medium for expressing the declarative knowledge in artificial intelligence systems [81];
3. Values, for example the desire for a simple theory or solution as exemplified in the principle known as Occam’s Razor;
4. Exemplars or Shared Examples, which are defined as the concrete problem-solution networks encountered by students of scientific disciplines in the course of their training, education and research apprenticeship (through the solving of textbook exercises, exams, and laboratory experiments) and by scientific practitioners during their independent research careers.

A particular set of assumptions upon which several different design methods may be based, is often referred to as a methodological design paradigm. A methodological design paradigm may be, like models within a Kuhnian paradigm, metaphysical in origin, or purely heuristic in nature. Similar to the role of a Kuhnian paradigm in the context of scientific discovery, a design paradigm serves as a framework or starting point for the solution of design problems. It is, thus, fundamentally an abstract prescriptive model of the design process that serves as a useful scheme for constructing practical design methods, procedures, and (computational) tools for conducting design [146]. A design method does not constitute a design paradigm. By definition, methods based upon the same paradigm are equivalent in the sense that they share the same set of possible solutions. This set consists of all solutions to the problem except those that violate any of the assumptions that constitute the paradigm. Hence, a given design method may be regarded as concrete and a practical embodiment of a design paradigm; it is an explicitly prescribed set of rules which can be followed by the designer in order to produce a design. A paradigm, according to the definition, will provide a framework or scheme for one or more design methods, just as it may serve as a framework or scheme for descriptive and automated tools.

Various schools of thought may become associated with a design paradigm. For example, in hardware logic design ('gate-level' design), the so called 'Eastern School' (a 'Naturalist' methodology) favored the use of block diagram in designing basic circuits, while the 'Western' School (a 'Formalistic' methodology) advocated the use of Boolean algebra [146]. Another example, in structural engineering of bridge design, concerns the debate between advocates of mathematical analysis of structural forces as subordinate to the development of structural form, and the approach that sophisticated analysis of the structural forces has priority over (and is determined by) structural form [10, 146].

2.4.2 DESIGN PARADIGMS

There are two major approaches to increasing our understanding of design disciplines that lack sound scientific theories; case studies (the counterpart of exemplars or shared examples) and models (the counterpart of a disciplinary matrix). The case studies approach was prevalent in such disciplines as psychology, prior to the establishment of the experimental method. This technique is also predominant in engineering design which relies mostly on the situation interpretation. The second approach is to use a model to define and understand the design process. Various perspectives and models need to be considered in order to gain a better understanding of the design process. Although there is no single model that can furnish a perfect definition of the design process, models provide us with powerful tools to explain and understand the design process. Models can be classified into five major types of paradigms: Analysis-Synthesis-Evaluation (ASE), Case-Based, Cognitive, Algorithmic and Artificial Intelligence. Following is a review of each of these paradigms. The interested reader may refer to Dasgupta [146] for discussions of these issues.

2.4.3 THE ANALYSIS-SYNTHESIS-EVALUATION (ASE) DESIGN PARADIGM

The ASE design paradigm is a very widely believed paradigm in the engineering discipline. Three basic phases of design described by [20, 7, 51, 70] are analysis, synthesis and evaluation. Analysis is concerned with defining and understanding what must be translated by the designer to an explicit statement of functional requirements (goals). Synthesis is involved with finding the solutions among the feasible alternatives. Evaluation is concerned with assessing the validity of the solutions relative to the original functional requirements [20]. In general, several instances of these three phases may be required in order to progress from a more abstract level to a more concrete level in the design process. A general model of design can be visualized as a feedback loop of synthesis, analysis and evaluation. The ASE model of design process is inherently iterative; the designer repeatedly goes back to refine and improve the design until it satisfies the requirements. Analysis and synthesis are on the forward path of the design loop, while the evaluation process is on the backward path, verifying the synthesized solutions [99]. A cycle is iterated so that the solution is revised and improved by reexamining the analysis. It has been argued that these three phases of the design, which are imperative for any design, irrespective of domain, form a framework for planning and organizing design activity.

Figure 2.8 depicts a more comprehensive version of a commonly used model of product development and design process. The design activity is viewed as part of the total product development process. Engineering a product involves several stages [109]: The first stage involves a market survey for potential products. This is followed by the conceptualization stage, where a product is conceived either as the result of a need or motivated by a potential profit. In the research and development stage, the information needed for the design of the product is developed. The design stage involves configuring the product based on several constraints. This is followed by the manufacturing process which yields the actual product (often preceded by developing a prototype). The product is then tested for quality in the testing stage and marketed in the marketing stage. The maintenance stage of the product is provided as a service by most organizations. The above process is iterative; for example, problems may arise during manufacturing and the product may have to be redesigned.

The process of solving a typical design problem involves various stages: problem identification, analysis, decomposition, synthesis, testing, evaluation and detailed design. The first task is concerned with identifying the problem (often fuzzy in nature), resource limitation, target technology, etc. Analysis involves listing the requirements and performance specifications, as well as specifying the constraints and objectives. Synthesis is the process of selecting components to form a system that meets design objectives while satisfying constraints that govern the selection. The components may themselves be complex entities which need to be synthesized first. Decomposition is often resorted to as a means for synthesizing the artifact into smaller and smaller components [17]. The artifact is decomposed into a hierarchical assembly of systems and subsystems until terminating in a functional or physical

attribute. In such a case, components, individually and through interaction with each other, meet the design goals. The testing stage involves the response of the system to external effects. This is determined by using an appropriate model of the system (such as stress and thermodynamic analysis) to check the feasibility of a design. Evaluation of such a design involves critiquing the solutions relative to the goals and selecting among alternatives. Traditionally, evaluation criteria have been represented either as production rules in production rule-based systems or as constraint rules in blackboard-based systems [27]. Other measures of performance for engineering design were constructed with the help of the theory of fuzzy sets [23, 26, 123]. The scope of these evaluators has been restricted to testing the validity of a solution rather than its degree of acceptability. In order to achieve these goals, design critiquing which involves evaluating a design in terms of its effectiveness in satisfying a set of design objectives and constraints, was recently proposed [90, 56]. Detailed design involves the determination and evaluation of several preliminary geometrical layouts of designs. Various components of the design are refined so that all applicable specifications are satisfied. All seven stages are inextricably intertwined and are not distinct phases in the process of design. Essentially, there are three possibilities for feedback-edges from testing back to analysis and synthesis, and from evaluation back to problem identification. In the first case, the design (or form) fails to satisfy one or more of the requirements. The design must then be modified by returning to the synthesis stage. In the second case, new requirements (or constraints) emerge during testing, and the design fails to satisfy one or more of them. The new requirements must then be integrated with the 'current' requirements and further analysis must be done. The outer cycle (Figure 2.8) demonstrates that the evaluated solution might revise the perceived needs.

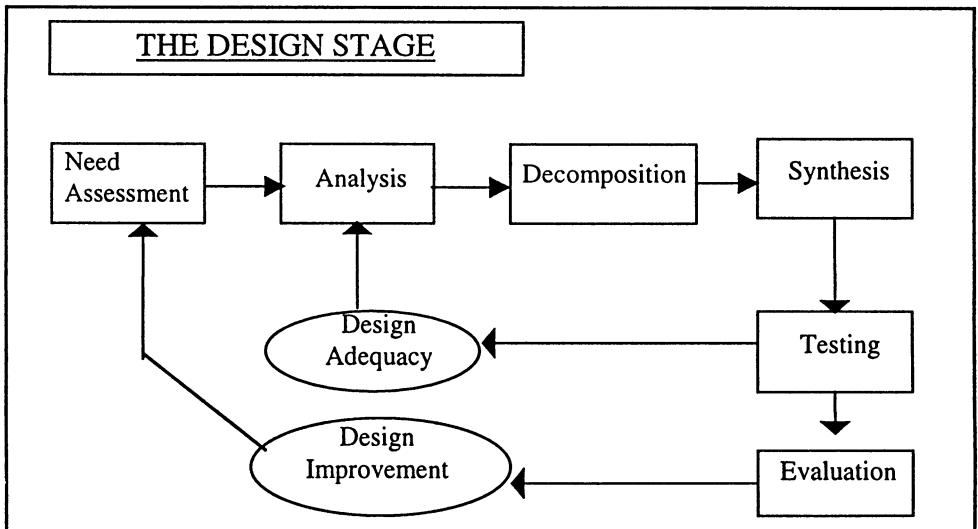
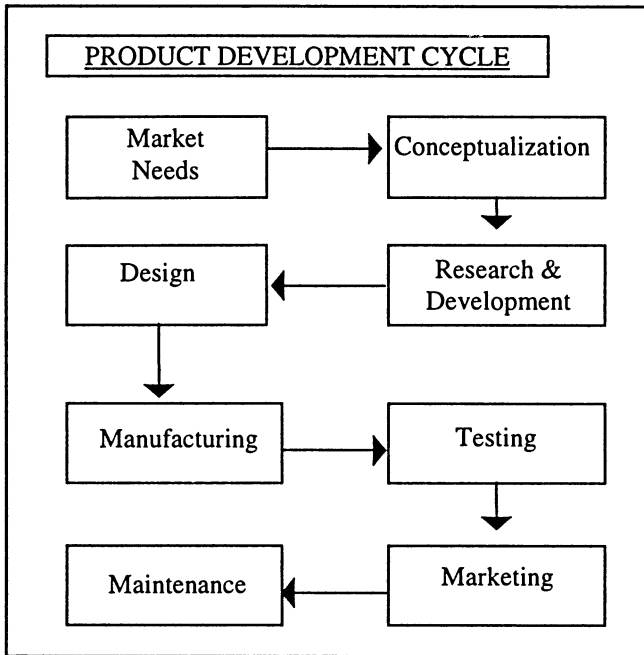


Figure 2.8 Comprehensive Model of the Design Process

Another view (or style) of the above design stages, which is popular in many European countries, is described by [85]. This design model involves the following stages: Clarification (which is similar to the above first two stages); Conceptual Design (which is similar to the above three later stages); Embodiment, where several preliminary geometrical layouts of designs are obtained and evaluated; and Detailed Design (same as above).

The Conceptual Design activity, constitutes the major part of the design process. The stage of conceptual design considerably determines the direction, flexibility and bounds of the design. It has been shown [121] that the conceptual design stage constitutes only 3% of the total product resource costs (research and development), while it determines almost 50% of the product's features including performance, manufacturability, production costs and other concurrent engineering factors. Hence, designers should carefully devise efficient design synthesis and evaluation tools.

The preceding section described mainly the descriptive role of the ASE paradigm. However, from the perspective of design paradigms a more interesting issue is whether the ASE can serve as a basis for developing designs. Alexander [4] devised a method which includes a stage of extensive, detailed and comprehensive analysis of requirements, then one or more stages of synthesis. Sriram and Cheong [109] have provided a brief description of an industrial product, Supercritical Fluid Chromatography, which is based largely on the ASE paradigm. Finally, traditional configuration design procedures of Flexible Manufacturing Systems (FMS) comply with this paradigm [29].

Several methods within the ASE design paradigm have recently evolved from the same foundation of competitiveness in terms of achieving high-quality and low-cost products (these include Concurrent Engineering, Design For Manufacture, Quality Function Deployment and Robust Designs techniques, see [100]). While it is beyond the intention of this book to go into these methods in great detail, an indication of the relation to the ASE design paradigm is given. The most significant of these methods is Quality Function Deployment (QFD), developed in Japan in the 1970s, and popularized by the automobile industry. QFD can be (roughly) described as a four-phase approach to design [74]:

1. Customer requirements planning - translates customer expectations ('the voice of the customer') in the form of market research, competitor analysis and technological forecasts into the desired and specific product characteristics;
2. Product specifications - converts the customer requirements plan for the finished product into its components and the characteristics demanded;
3. Process and quality control plans - identify design and process parameters critical to the achievement of the requirements;
4. Process sheets (derived from the process and quality control plans) - are the instructions to the operator.

Thus, interpreting the QFD process in the terminology of the ASE design paradigm shows that Steps 1-2 constitute an analysis phase, whereas Step 3 constitutes a synthesis phase. The design process style that is invoked in the QFD process is a top-

down method (explicitly defined in later sections).

As product designs tend to become conceptually static, QFD will tend to become a more powerful method. It can also be used as a guideline for incremental redesign activity (recall Section 2.3.2). If, however, the design implementation is in the start-up growth stage (as a consequence of innovative redesign, for example), and the customer has yet to experience the benefits of these changes other methods may be invoked.

Although the ASE paradigm is a very widely believed design paradigm in the engineering disciplines, it bears several problems: First, with the explicit ordering of the three stages [22]. Second, with the division of the cognitive activities of analysis and synthesis [95, 114]. Third, with its preclusion of the role of the designer's viewpoint and system of beliefs (or a priori conceptual model) in the process of design. However, if a design problem is well-structured, the design space is sufficiently small (see Section 2.2.4), and the designer uses conceptual models (that is, the overall design of the artifact is known beforehand), then the ASE may be an appropriate paradigm (both descriptive and prescriptive).

2.4.4 CASE-BASED DESIGN PARADIGM

In contrast to other design domains, such as software engineering and circuit design [113], a simple and obvious correspondence between specific functional requirements of the artifact and individual components in the design does not usually exist. Due to the tightly coupled and interacting nature of mechanical designs, reasoning from prior design cases is proving to be a suitable design methodology as opposed to direct 'decompose and recombine' (or 'generate and test') strategies that have successfully been utilized in very-large-scale integration (VLSI) design [116, 111]. Cases are the primary way in which engineering students are taught to design. This is because there are no general algorithms for design. The designer activity is a consequence of his experience and training, much of which is based on previous exposure to similar design problems. This is particularly true in engineering design [85; 39]. Even when a novice engineer joins a design project, an important part of the engineer's training involves going through the design records of previous projects. Cased-based problem solving is based on the premise that a design (or a machine) problem solver makes use of experiences (cases) in solving new problems instead of solving every new problem from scratch [54]. Design cases reflect good design principles, such as function sharing [113] and incorporate decisions that take advantage of, or compensate for, incidental component interactions. Lansdown [65] argues that "innovation arises from incremental modification of existing 'tried and true' ideas rather than entirely new approaches ...the transformation from initial to final description is continuous and design is more like fine-tuning a set of already working ideas rather than inventing something new, although the results might not resemble anything previously imagined." Coyne et al. [20] use the similar term 'prototype model': "A prototype typifies, or exemplifies, a class of designs, and thus serves as a generic design ...a description of a class of designs also may be

prototyped or knowledge or rules may even constitute a prototype." A particular design can then be instantiated or exemplified from the class (prototype) of designs.

Coyne et al. [20] classify the case-based paradigm into three activities: creation, modification and adaptation. Creation is concerned with incorporating requirements to create new prototypes. Modifications is concerned with developing a working design from a particular category of cases. Adaptation is concerned with extending the boundaries of the class of cases. Pugh and Morley [89] have conducted extensive design process research by interviewing successful design teams in British industry. The research indicates that embodiment design (models, prototypes etc.) may be produced very early in the process of design, both in incremental and innovative design activities (see III-A).

2.4.5 THE COGNITIVE DESIGN PARADIGM

A cognitive model is representative of how people perform a mental task or activity and the interrelationships of active intelligent human designers with computerized tools such as computer aided drafting systems. Protocol studies on how engineers design were conducted by several researchers [2, 37, 119]. In these studies, designers were given problems to solve in a specified amount of time and were asked to think aloud (protocol analysis as a technique to study problem solving behavior is discussed and used extensively in [79]). Based on these studies, the researchers have formulated several models of the design process. For example, Ullman et al. propose a model of the mechanical design process called the Task/Episode Accumulation process. Their model views the design process as consisting of the Conceptual Design, the Layout Design, the Detail Design, and the Catalog Selection stages. A set of ten operators are used to accomplish these stages. They also observed that designers normally pursue only a single alternative, rather than considering multiple alternatives. Sriram and Cheong [109] indicate (based on case studies) that while designers may have difficulty retaining several alternatives in their memory, the designers feel that tools that will aid them to pursue various choices would produce more innovative designs. Many of the features incorporated into CAE (computer aided engineering) tools were influenced by the cognitive studies. CAE tools have been utilized for diverse domains. Paper path handling, air cylinders, buildings and circuits are few examples of domain dependent/independent frameworks developed in the mid 80's. These systems used hierarchical refinements and constraint propagation problem solving strategies.

Throughout the spectrum of the design process categories, the process of creation or ideation often follows a definite pattern [42]:

1. Preparation - defining the situation and gathering facts;
2. Frustration - struggling against mental blocks;
3. Illumination - a sudden spark of insight;
4. Evaluation and execution - assessing alternatives and implementing the optimal choice (contingent to the designer's world view).

The following attributes are identified as common elements of creativity:

1. Capacity for intuitive perception: the recognition of associations and similarities among objects and concepts;
2. Concern for implications, meanings, and significance;
3. Ability to think imaginatively without regard for practicalities;
4. Open-mindedness toward, for change, improvement, and new ideas rather than rehashing old techniques and traditions. The creative designer is warned of the cost of spending too much effort researching solutions to similar problems of the past.

A number of techniques are available which appear to animate the creative process (a prescriptive view of the cognitive paradigm). Examples are: the trigger word method in which a designer asks himself a series of active questions. The checklist method [82] that relies on a number of questions on modifications. The morphological method that analyzes the problem and determines the independent parameters involved which are then listed on a grid and evaluated systematically. The Gordon technique that attempts to identify the fundamental concepts underlying a given situation rather than emphasizing the obvious characteristics. The brainstorming technique which refers to the spontaneous generation of ideas by a diverse group of individuals, some of whom may know little about the particulars of the problem.

2.4.6 THE CREATIVE DESIGN PARADIGM AND THE SIT METHOD

We briefly present the creative design paradigm and the Structured Inventive Thinking (SIT) method, which efficiently implements and enhance creative problem solving in engineering design. For details see references [150, 151].

The SIT method is a three-step procedure: problem reformulation; general search strategy selection; and an application of idea provoking techniques. The most innovative part of the method is the problem reformulation stage. The given problem is modified through the application of clear, objectively defined and statistically proven set of sufficient conditions for creative solutions. Extensive empirical cases that were analyzed proved with high statistical confidence that the method leads the designer to creative solutions.

The cases also prove the correlation of the SIT method and classical psychological tests of creativity, in two aspects. Students who are creative according to the psychological tests are also achieving better results with the SIT method. Most important, teaching the SIT method enhance creativity in students and engineers.

The SIT theory states that if an idea for a solution of a technological problem satisfies two sufficient conditions, that idea will be deemed creative by field experts. Using SIT, the problem solver first reformulates the problem by changing the goal from 'find a solution' to 'find a solution that satisfies the conditions'. The problem solver then proceeds to the process of searching the solution. At this stage the

designer selects one of two general solution strategies, each leading to a different set of idea provoking techniques.

The SIT method uses ideas developed initially by Altshuller who examined thousands of inventions and patents from which he extracted 39 properties that characterize creative solutions. Based on his findings the TRIZ method was developed.

SIT differs from TRIZ in some fundamental aspects, especially in the fact that in SIT only two fundamental principles lead the paradigm, and these principles are clearly and rigorously formulated as the sufficient conditions. Reformulating the problem through the sufficient conditions generates a well-defined and clear criterion leading toward testing a candidate solution. Another important difference between the two methods is that SIT applies a minimal set of techniques, so that after some training the SIT process can become second nature to the problem solver.

SIT is used in many companies including Ford Motor Company in the US (trained by Dr. Sickafus) and many Israeli hi-tech companies. The method is taught as a full credited academic course in Tel-Aviv University, and in the National University of Singapore. SIT practitioners have attained many creative solutions.

Formal Expression of Sufficient Conditions

We start with needed notation related to a situation in which a problem is described in terms of a given (existing) technological system that suffers from (known) undesired effects.

S_i	the given system in the problem state (I for input)
S_o	the system in the solution state (o for output)
$N(s)$	the neighborhood of a system S (the collection of objects which are not an integral part of the system but can be found in the system's proximity or have special affinity to that system)
$O(S)$	the collection of object types from which the system S is composed. Each object stands for the single technological concept that underlies its functioning in the system.
UDE	the collection of variables which contribute, directly or indirectly to the undesired effects, that appear in the problem description
$f^+(y,x)$	y is an increasing function of x , when all other variables remain constant.
$f^-(y,x)$	y is a decreasing function of x , when all other variables remain constant.
$f^0(y,x)$	the value of y is independent of the value of x

If $x, y \in UDE$, $f^+(y,x)$ is called a problem characteristic function.

Using these notations and definitions the two sufficient conditions can be expressed as follows:

The *Closed World (CW)* condition:

$$O(S_o) \cup N(S_o) \subseteq O(S_i) \cup N(S_i) \quad (1)$$

The *Qualitative Change in Problem Characteristic (QC)* condition:

$$\text{For } x, y \in UDE, [f^+(y, x)]_{S_i} \wedge [f^0(y, x) \vee f^-(y, x)]_{S_o} \quad (2)$$

The expression for the *closed world condition* means that no new object can be added to the system, unless it is a neighborhood object, but objects can be removed from the system. Since $O(S)$ stand for object types and not the objects themselves, more objects of the same type are allowed to be introduced into the system (e.g. add more wheels to a car).

The expression for the *qualitative change in problem characteristic condition* means that a problem characteristic needs to change from an increasing function to either a decreasing or an unchanging function.

The sufficient conditions were developed through an empirical survey of numerous engineering problems and their corresponding routine and creative solutions. Once the conditions were extracted an explanation for the rational behind them may have been induced: Commonly routine design problem solving processes begin with an attempt to tune parameters, and when this fails to produce the desired results, engineers turn to searching alternative technological concepts. The QC condition makes parameter tuning ineffective, and the CW condition does not allow a replacement of existing concepts. Routine processes thus fail to produce the desired results, and the problem solver is forced to resort to more creative processes.

Description of the SIT Mechanism

The SIT mechanism comprises three main steps: problem reformulation through the sufficient conditions; selection of a general thinking strategy; and selection and application of a relevant idea provoking technique.

(i) Problem Reformulation:

At this stage the problem solver sets the target for the problem-solving task using the two sufficient conditions. The CW condition is added to current constraints, while the QC condition changes the goal: instead of the initial (and natural) requirement to decrease the level of an undesired effect, the problem solver is guided to qualitatively change a mathematical relation between any two problem related variables (problem characteristics).

Technically at this stage the user forms a list of system objects, a list of system neighborhood objects, and a list of problem characteristic variables. The problem solving task is defined as follows: Find a solution in which at least one of the defined

increasing functions will become decreasing or unchanging subject to the constraint that the solution will incorporate only elements of the given system and its neighborhood that appear in the relevant lists.

(ii) Strategy Selection:

The framework of the sufficient conditions naturally gives rise to two thinking strategies. A candidate solution is composed of three elements: the desired *physical end state* - deduced from the QC condition, the *objects to be modified*, and the *required modification*. The CW condition confines the objects to be modified only to existing ones, and thus significantly narrows the search space. Two scenarios are possible at this stage: The problem solver can deduce a required physical end state from the QC condition. This situation commonly occurs when the desired End State can be achieved through a simple physical operation that will not interfere with other operations required from the system.

The problem solver cannot conceive a desired physical end state, or the state he can think of contradicts other fundamental requirements from the system. These two scenarios define the two possible strategies. Following the first strategy, the problem solver first formulates a conceptual solution: a simple operation that once added to the system, the QC condition is guaranteed to be satisfied. He then proceeds to find an existing object that will carry out the desired operation. Consider for example the problem of testing acid liquid material. In this case, the tested material causes corrosion of the vessel that holds the material. In the course of the solution of the material testing problem, the problem solver can think of the idea to physically separate the acidic liquid from the vessel - an idea that guarantees the satisfaction of the QC condition. He then selects an existing object: the tested samples to carry out this operation. This strategy is called the *extension strategy* to indicate the fact that the system is temporarily extended through the addition of an imaginary object that will carry out the new operation.

If the problem solver reaches at this point the second situation (that is he cannot conceive of a desired end state that would guarantee the satisfaction of the QC condition) he can follow a different strategy: through a trial and error process he tries different possible modifications to existing objects until at some point hopefully he hits a state where the QC condition is satisfied (the CW condition is guaranteed to be satisfied since none of the tried modifications violates it). This strategy is called the *restructuring strategy* to indicate the fact that in the trial and error process the problem solver changes the structure of existing objects and their organization.

The problem solver is guided to select the extension strategy if he can conceive a conceptual solution, and to select the restructuring strategy otherwise. The actual significance of selecting a thinking strategy lies in the application of a different set of idea provoking techniques for each strategy. If the extension strategy was selected the user is directed to apply either *unification* or *multiplication*, if the restructuring strategy is selected the user is directed to apply either *division*, or *increasing variability*. The extension techniques help the problem solver identify an existing object that will carry out the new operation, while the restructuring techniques help

him increase the degrees of freedom of possible changes to the system.

(iii) Idea Provoking Techniques:

Idea provoking is the final stage of the method. Their main role is to free the problem solver from fixated mental states.

The Unification Technique. The unification technique helps the problem solver identify a system or neighborhood objects that will carry out the operation defined in the conceptual solution. Applying the technique is a four-step process:

- (1) Formulate the needed operation.
- (2) Form a list of all main system and neighborhood objects.
- (3) Select an object from the list and complete the following sentence: The *selected object* will carry out the *operation*.
- (4) Determine the necessary modifications of the selected object, so that it can carry out the desired operation.

Example - the four steps applied to the material testing problem:

- (1) The needed operation: to separate the acidic liquid from the vessel.
- (2) A list of objects: vessel, samples, and acidic liquid.
- (3) Selected object: samples. The samples will separate the acidic liquid from the vessel
- (4) Required modification: the shape of the samples will change so that it can contain liquid.

The Multiplication Technique. The purpose of this technique and its first 2 steps (out of four) are identical to the Unification technique. The last two steps are listed below:

- (3) Select an object from the list and complete the following sentence: The selected object will multiply. The new copy (or copies) of the of this object will carry out the operation.
- (4) Determine the necessary modifications of the new copy (or copies) of the selected object, so that it can carry out the desired operation.

The Division Technique. Being a restructuring strategy technique the purpose of the division technique is to help the user identify new degrees of freedom for modifying and reorganizing system objects. It is a three-step process:

- (1) Form a list of system objects.
- (2) Select an object from the list and complete the following sentence: The object will be divided to its more basic elements/to smaller parts of the same part/in a random way (select one option from the three).
- (3) Search for meaning: try to use the new degrees of freedom to create a state in which the QC condition is satisfied: different parts in different locations, different order of parts etc.

The Increasing Variability Technique. This is another important technique to aid the problem solver in the creation of new degrees of freedom and new ways to solve the problem. It is a four- step process:

- (1) Form a list of system objects.
- (2) Select an object.
- (3) Select two parameters W and Z that are currently not related, that is W is not a function of Z (a new degree of freedom will be the type of relation between them).
- (4) Search for meaning: try to use the new degrees of freedom to create a state in which the QC condition is satisfied.

Examples for the Application of SIT

This section will demonstrate how SIT is used to find creative solutions to technological problems. Each example will include problem description, a set of routine solutions (most common responses of engineers), detailed description of how SIT is used to solve the problem, and a description of a creative solution - the output of the SIT process. It is important to note that common SIT applications consist in many trial and error processes (different strategies, different techniques, and different application of the techniques), however the description below does not reflect that important nature of SIT application. For sake of brevity we describe in each example only the direct path to the solution.

EXAMPLE 1 - CORN IN A PIPE

A curved steel pipe is one of the components of a corn grain processing plant (as described in Figure 1). The pipe's function is to conduct the flow of air with the corn grains. The problem is the grains' impact on the pipe at the bend, which erodes the pipe wall. The air speed cannot be reduced since this will reduce the plant's capacity.

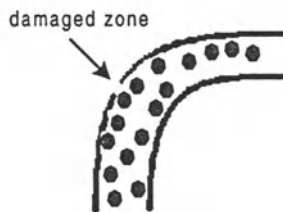


Figure 1 The Corn in the Pipe Problem

Routine ideas:

- (1) To strengthen the pipe at the erosion zone (change the material, make it thicker).
- (2) To make the curved part of the pipe from a different piece that can be easily replaced.

(3) To coat the pipe with a protective layer that will be replaced from time to time.

SIT Step 1 - problem reformulation:

- (1) List of UDE parameters: cost of production, erosion rate, grain flux, grain hardness.
- (2) List of system and neighborhood objects: pipe, grain, and airflow.
- (3) The reformulated problem: make erosion rate unrelated to/decreasing function of grain flux, don't add any new object to: pipe, grain and airflow.

SIT Step 2 - strategy selection:

Since a conceptual solution can be conceived - to separate the grains from the pipe the extension strategy is selected.

SIT Step 3 - select and apply an idea provoking technique (unification or multiplication):

Unification was selected, application of the technique:

- (1) Formulate the needed operation: to separate the grains from the pipe
- (2) Form a list of all main system and neighborhood objects: grains, pipe, airflow
- (3) Select an object from the list (grain) and complete the following sentence: *The grain will separate the grain from the pipe.*
- (4) Determine the necessary modifications of the selected object, so that it can carry out the desired operation: grains should stick to the curved part of the pipe

The solution: The geometry of the curved area of the pipe will change as to create a pocket that will enable the grains to accumulate there. This will protect the pipe from the damage of grains' impact (see Figure 2).



Figure 2 The Solution to the Corn in the Pipe Problem

EXAMPLE 2 - DERAILING DETECTION DEVICE

The braking system of trains includes a pipe that passes along the train, in which the air is at a pressure of 5 atmospheres. When the pressure drops, the train stops. Under emergency conditions (such as derailling), the air must be released very quickly. To ensure fast enough release of the air, it should exit through an opening that is at least 10 cm². During normal operating conditions, this opening should be closed with a

stopper. The air pressure itself should release the stopper.

A new derailing detector has been developed. The idea is that in normal operation, the stopper is held in place by the derailing detector, and when derailing occurs the detector stops exerting force on the stopper and it is released. The problem is that the derailing detector can exert only 0.5 Kgf, not enough to balance the 50 Kgf applied by the internal pressure (see Figure 3).

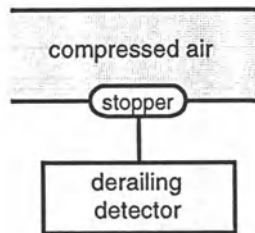


Figure 3 The Derailing Detector System

Routine ideas:

- (1) To use a lever.
- (2) To add more derailing detectors, each will support a smaller stopper.
- (3) To squeeze the stopper in its place so that friction will carry out some of the load.

SIT Step 1 - problem reformulation:

- (1) List of UDE parameters: probability of false alarm, probability of premature stopper opening, load on the derailing detector, air pressure, and stopper area.
- (2) List of system and neighborhood objects: pipe, air, stopper, and derailing detector.
- (3) The reformulated problem: make load on the derailing detector unrelated to/decreasing function of air pressure, don't add any new object to: pipe, air, stopper, and derailing detector.

SIT Step 2 - strategy selection:

Since a conceptual solution can be conceived - to exert on the stopper a force that is identical and in opposite direction to the force exerted by air pressure, the extension strategy was selected.

SIT Step 3 - select and apply an idea provoking technique (unification or multiplication):

Multiplication was selected, application of the technique:

- (1) Formulate the needed operation: to exert on the stopper a force that is identical and in opposite direction to the force exerted by air pressure.
- (2) Form a list of all main system and neighborhood objects: pipe, air, stopper, and derailing detector.
- (3) Select an object from the list (stopper) and complete the following sentence: The

stopper will be multiplied. The new copy (or copies) of this object will *exert on the stopper a force that is identical and in opposite direction to the force exerted by air pressure*.

(4) Determine the necessary modifications of the new copies of the selected object, so that it can carry out the desired operation: The new stopper should be slightly smaller than the original one so that the derailing detector will still have to carry some load.

The solution: The new stopper will be mounted exactly above the original one, they will be connected through a thin wire (see Figure 4).

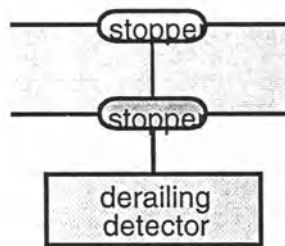


Figure 4 The Solution to the Derailing Detector Problem

EXAMPLE 3 - THE TUMOR PROBLEM

Suppose you are a doctor faced with a patient who has a malignant, inoperable tumor in his stomach. Unless the tumor is destroyed the patient will die. There is a ray that can be used to destroy the tumor. If the rays are directed at the tumor at sufficiently high intensity, the tumor will be destroyed. Unfortunately, at this intensity, the healthy tissue that the rays pass through on the way to the tumor will also be destroyed. At lower intensities, the rays are harmless to the healthy tissue but they will not affect the tumor.

Routine ideas:

(1) In this problem the most common ideas either violate problem definition (for example to try to operate although the problem text explicitly states that the tumor is inoperable) or suggest alternative treatment such as chemotherapy.

SIT Step 1 - problem reformulation:

(1) List of UDE parameters: probability of patient's death, damage to healthy tissues, and rays intensity.

(2) List of system and neighborhood objects: rays, tumor, and healthy tissues.

(3) The reformulated problem: make damage to healthy tissues unrelated to/decreasing function of rays intensity, don't add any new object to rays, tumor, and health tissues.

SIT Step 2 - strategy selection:

Since a conceptual solution cannot be conceived the restructuring strategy is selected.

SIT Step 3 - select and apply an idea provoking technique (division or increasing variability)

Division was selected, application of the technique:

(1) Form a list of system objects: rays (this is the only system object).

(2) Select an object from the list (rays) and complete the following sentence: The rays will be divided *to smaller parts of the same type*.

(3) Search for meaning: try to use the new degrees of freedom to create a state in which the QC condition is satisfied (different parts in different locations, different order of parts etc.): The smaller rays will be directed at the tumor from different angles.

The solution: To direct a few weak beams at the tumor from different angles, so that they converge at the tumor and develop sufficient intensity there to destroy the tumor. The QC condition was satisfied since it is possible to increase the ray intensity at the tumor by adding more weak rays without affecting healthy tissues through which the rays pass.

EXAMPLE 4 - SOLID FUEL ROCKET ENGINE

One of the problems that designers of solid-fuel rocket engines (Figure 5) faced was the necessity of achieving a constant thrust from the engines. The solid-fuel rocket engine has the shape of a hollow cylinder burning in an internal envelope. The problem with such geometry is that the thrust is not constant owing to a change in the area of the internal envelope (the radius increases). When the internal combustion area increases, the thrust increases.

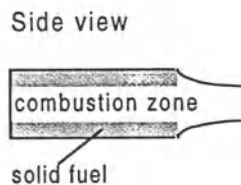


Figure 5 Side View of A Rocket Engine

Routine ideas:

(1) A new parametric design: the dimensions of the cylinder are changed so that it will become longer and narrower. These changes maintain its total volume and combustion area, but since the difference between initial and final radius is smaller, the variance is smaller.

(2) "Cigar burning" - a cylinder burning in its base.

SIT Step 1 - problem reformulation:

- (1) List of UDE parameters: energy waste, uneven thrust, thrust increase, burning area increase, perimeter increase.
- (2) List of system and neighborhood objects: solid fuel, rocket, and thrust.
- (3) The reformulated problem: make burning area unrelated to/decreasing function of perimeter, do not add any new object to solid fuel, rocket, and thrust.

SIT Step 2 - strategy selection:

Since a conceptual solution cannot be conceived the restructuring strategy is selected.

SIT Step 3 - select and apply an idea provoking technique (division or increasing variability).

Increasing variability was selected, application of the technique:

- (1) Form a list of system objects: solid fuel, rocket, and thrust.
- (2) Select an object: solid fuel.
- (3) Select two parameters W and Z that are currently not related, that is W is not a function of Z (a new degree of freedom will be the type of relation between them): Z - cross section shape; W - combustion progression.
- (4) Search for meaning: try to use the new degrees of freedom to create a state in which the QC condition is satisfied: The shape of the cross section will change through the combustion progression from a very complicated winding shape to a pure circle.

The solution: the shape of the cross-section is such that it maintains a constant perimeter while combustion progresses. The cross-section changes from a complex shape to a simple circle, thus, although the average radius increases, the perimeter remains constant. Figure 6 demonstrates the idea. This solution preserves the initial concept of a hollow shape burning in the internal envelope thus complying with the closed world condition. Since the variance in thrust is constantly zero, totally independent of the difference between initial and final radius, the solution satisfies the QC condition as well. Note that, at its time, this solution was a breakthrough in solid fuel engines. In our workshops we see students using the sufficient conditions finding this solution quite quickly.

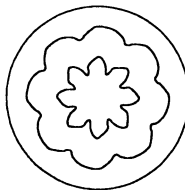


Figure 6 The New Inner Envelope as Combustion Progresses

Conclusion

This section presented a set of objective sufficient conditions for creative solutions,

and a three-step method that structures the search toward obtaining engineering design solutions.

The application of the sufficient conditions to a specific problem modifies the given problem definition: The QC condition changes the goal of the search and the CW condition confines the search space. By applying the sufficient conditions, the creative problem solver actually solves a different problem than his non-creative companion, an important factor in finding a different, sometimes surprising solution. Note that the QC condition is related to the functional decoupling requirement in Nam Suh's paradigm (see Section 2.5.1).

The detailed extensive study of the validity of the sufficient conditions and the SIT method can be found in the references.

2.4.7 THE ALGORITHMIC DESIGN PARADIGM

The algorithmic design paradigm is the focus of a considerable amount of current and past research of design automation. The algorithmic design paradigm views the design process as the execution of an effective domain-specific procedure that yields a satisfying design solution (relative to a given initial requirements) in a finite number of steps. The main premise of this paradigm is that the requirements are well-defined and there are precisely defined criteria for determining whether or not a design meets the requirements. That is, the notion that design problems are well-structured (see Section 2.2.4).

There exist a number of instances of the algorithmic paradigm which serve to optimize complex systems: exhaustive search, rapid search and mathematical programming techniques. Within the algorithmic paradigm, exhaustive search is a lengthy process resulting in global optimization within the field of inquiry. A style is defined as a set of attributes that enables the designer to discriminate between one set of artifacts and another in the same group. A style, therefore, represents certain search modes (design process styles) on the part of the design algorithm, the result of which is the nature of the final design [see also 103]. A number of search modes exist for algorithmic paradigms: breadth-first, depth-first, greedy method, branch and bound, dynamic programming and so on.

The price of global optimization through an exhaustive search of alternatives is tremendous. The alternative to an exhaustive search is rapid search, where a set of simple but arbitrary guidelines are adopted to limit the search space. For example, in serial optimization, as each stage is optimized (e.g. selecting the types of materials and method of manufacture), the selections at the subsequent stages are evaluated conditionally with the assumption that the preceding choices hold. The algorithm proceeds in this way throughout the series of stages. Serial-optimization may be the most widely used design style in conscious human decision-making. In term of effort, it is clearly superior to exhaustive search. The greatest disadvantage of any rapid search method lies in the questionable proximity to the global optimum; the rapid-search algorithms use arbitrary guidelines for optimization. Notwithstanding the global optimization potential, many instances within the algorithmic design paradigm

produce satisfying rather than optimal solutions. To understand the difference between exhaustive and rapid search, consider the domain of arc welding design. Over the years, numerous researchers have studied various aspects of the welding process, such as understanding the underlying fluid mechanics, heat transfer, phase transformation and solid mechanics of the welding process. Attempts to incorporate them into a rapid search strategy often results in a nonsystematic, somewhat random method in which designers formulate a set of decisions that ultimately result in the welding process. Most decisions are made to optimize one aspect of the process, rather than the process as a whole. This often results in a suboptimal design of the welding process. A globally optimized welding process, through exhaustive search, may be achieved by requiring a complete and thorough understanding of the complex interactions among various aspects of the welding process. The difficulty with this approach is the global understanding required of an incredibly large data base.

Mathematical programming techniques are a recognized topic in many engineering design courses, and are discussed at length in texts on engineering design theory [101, 24]. Mathematical programming techniques can be used to identify the potential design configuration (e.g. the physical design of electronic circuits) by optimizing the configuration based on the functional requirements. In general, in these methods the solution to the problem is developed by solving the mathematical model consisting of an objective function that is to be optimized and a set of constraints representing the limitation of the resources.

In summary, although most interesting design problems are incomplete, open-ended and ill-structured (see Section 2.2.5), they may be decomposed into one or more well-structured components. In this case, the algorithmic paradigm may be successfully utilized to solve each of these well-structured sub-problems. Thus, the algorithmic paradigm may be considered as a tool that can support and be invoked by other paradigms [17, 22].

2.4.8 THE ARTIFICIAL INTELLIGENCE DESIGN PARADIGM

Artificial intelligence (AI) is the field that attempts to make computers perform tasks that usually require human intelligence. The AI design paradigm is based upon capturing the knowledge of a certain domain and using it to solve problems [34]. In order to automate a design process, a design system must be able to differentiate between various choices and determine the best path. The AI design paradigm views design as a problem-solving process of searching through a state-space, from an initial problem state to the goal state, where the states represent the design solutions. Transitions from one state to another are affected by applying one of a finite set of operators, based on the functional requirements (goals) and design constraints (constituting the domain specific knowledge) and meta-rules (constituting the domain independent knowledge). The design process involves representing much of their knowledge about the problem declaratively. Roughly speaking, declarative knowledge is encoded explicitly in the knowledge-base in the form of sentences in some language (usually in the form of IF condition THEN action), and procedural

knowledge (which typifies the algorithmic design paradigm) is manifested in algorithms.

The AI paradigm of design relies heavily on the Function-Structure-Behavior of an artifact and their inter connections through causality. People often refer to artifacts based on the functions they provide, and that existing structures could be combined into new ones, to achieve the desired function [30]. Bobrow [11] defines function as the relation between a goal of a human user and the behavior of the system. Structure is defined as the information about the interconnection of modules, organized either functionally - how the modules interact - or physically - how it's packaged. Behavior can be defined as the relationship between input from the environment and the output of affect the component usually interfaces to the environment. In summary, the proponents of the AI design paradigm claim that through a causal reasoning approach using the problem-solving process of searching and the basic physics behind behaviors, the structure and set of behaviors to achieve a given goal in as little time as possible can be accomplished.

The use of knowledge-based expert systems (KBES) has become common enough to understand their benefits in a problem such as this. An expert system is able to use previously defined rules and cases to choose through a new problem to solve it. This would enable an expert system to pick previous design structures and behaviors and match them to the design specifications. How a knowledge-based expert system should be constructed depends on where the problem lies on the analysis-synthesis spectrum [109]. In analysis (e.g., process diagnostics), the problem conditions are posed as parts of a solution description; the possible outcomes exist in the knowledge-based of a KBES. Essentially, the solution to these problems involves the identification of the solution path. In synthesis problem (or design) problems conditions are given in the form of properties that a solution must satisfy as a whole; an exact solution does not (normally) exist in the knowledge-base, but the inference mechanism can generate the solution by utilizing knowledge in the knowledge-base. Figure 2.9 depicts a multi-level framework for describing knowledge-based design as enunciated by [117]. KBESs are instances of automatic problem solvers that rely heavily on domain-specific heuristics, are also often called strong methods [80]. Knowledge-based expert systems provide the support for many of the automatic or computer-aided design systems developed in recent years, such as buildings design, circuit design, paper path handling and air cylinders [117; 97]. Sriram and Cheong [109] have identified the following tenets of computer-aided design systems:

1. Incorporate design plans, design knowledge and design constraints;
2. Deal with evolving specifications;
3. Display geometry and have the ability to associate constraints with geometrical entities (relating structure to behavior);
4. Provide access to distributed knowledge/database of devices;
5. View multiple alternatives simultaneously;
6. Provide access to manufacturability knowledge;
7. Generate assembly sequences automatically from geometry.

When less is known about the design task environment, domain-independent control strategies are more appropriately evoked. Problem solvers that rely heavily on domain-specific heuristics, are also often called weak methods [80]. Weak methods are used to effect and control the search through the state-space. The so-called means-ends analysis [79] lies at the center of these methods (see also Section 2.3.2). Means-ends analysis was employed, among others, by [77] in the domain of automatic program synthesis; and by [34] in designing room configurations.

The AI design paradigm may be combined with other design paradigms to establish a 'grand' problem solving strategy for the designer or design system. It is only very recently that the use of past cases is beginning to be recognized in the design automation literature [76, 32, 47]. The process of case-based design consists of the following steps that are iteratively applied as new sub-goals are generated during problem-solving [115]:

1. Development of a functional description through the use of qualitative relations explaining how the inputs and outputs are related;
2. Retrieval of cases which results with a set of design cases (or case parts) bearing similarity to a given collection of features;
3. Development of a synthesis strategy which describes how the various cases and case pieces will fit together to yield a working design;
4. Realization of the synthesis strategy at the physical level;
5. Verification of the design against the desired specifications through quantitative and qualitative simulation;
6. Debugging which involves the process of asking relevant questions and modifying them based on a causal explanation of the bug.

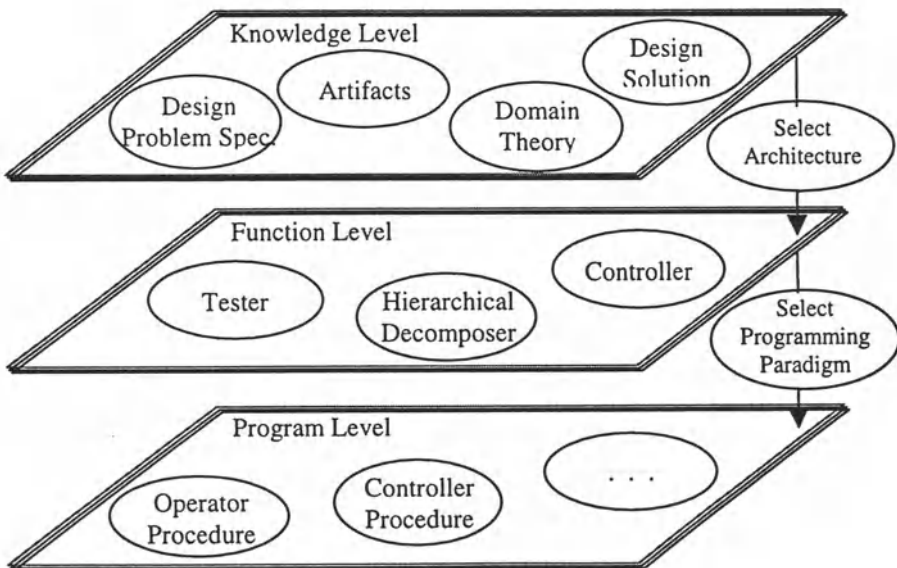


Figure 2.9 Multilevel Framework for Describing Knowledge-based System (adapted from [109])

Case-based problem solving has several advantages over knowledge-based expert systems: first, cases provide memories of past solutions, failures and repairs that have been used successfully. Secondly, from a knowledge acquisition view, there is experimental evidence that designers don't explicitly think in terms of rules [107]. Moreover, asking a designer to give examples of cases, is much easier than asking him to give a list of rules he uses to design. Finally, design case studies are readily available in the literature.

Simon pointed up [105] that the design process strategies (domain-dependent as well as domain-independent) "can affect not only the efficiency with which resources for designing are used, but also the nature (form style) of the final design as well." Conversely, the designer is likely to use some special features of the problem to identify one or a small number of styles that are evoked in the process of design (that is, a form style as a determinant of a control strategy or design process style). The main three types of design process styles [105] are bottom-up, top-down and meet-in-the-middle. Bottom-up design style involves starting with basic structures and combining these structures until the final form is accomplished. This design style incorporates search trees and optimal branch calculations. Top-down design starts with the final behavior required and sub-divides this behavior into smaller behaviors which are ultimately linked to components and their respective structures. This design process style is similar to bottom-up design in that it uses search trees and optimal branch calculations. Meet-in-the-middle design process style incorporates the previous two design process styles. Either top-down or bottom-up is chosen according to the difficulty encountered in the design process and the amount of previous information available.

To sum up, the AI design paradigm is very useful in solving tightly coupled, highly integrated and ill-structured design problems. However, when faced with an original design problem with no previous rules or past cases to help it, the expert system or case-based problem solver are incapable of original creativity. Thus the expert system or case-based problem solver are capable of helping in the design process but are not the solution to design automation. Moreover, the real problem with AI in design is in determining a way to make the computer program be creative in a manner that is manageable and not an NP hard solution. If the program just generates every possible solution, something the computer excels at, the amount of time generating and checking the solutions may be infinite. The design program must be able to generate a small selection of design solutions and choose between them for the ultimate choice. The human designer is able to do this quite easily but has an extensive language and other extraneous factors to assist in this process.

2.4.9 DESIGN AS A SOCIAL PROCESS

The ASE design paradigm (the "consensus" paradigm) describes the engineering design process as a sequence of activities leading to intermediate results. Moreover, many design theorist still insist that design theory requires universal methods analogous to universal methods used in the natural sciences. In contrast, it is the

contention of the *social constructivist* approach that the study of design and sociological studies should, and indeed can, benefit from one another [127].

According to the social constructivist approach, design as a social process involving designers, customers, and other participators consists of creating and refining a shared meaning of requirements and potential solutions through continual negotiations, discussions, clarifications, and evaluations [126]. From the social constructivist perspective, the consensus model overlooks or underplays the social factors involved in the design process. For example, negotiation can enter into all phases of the process. There is also no mention of the problems of sharing knowledge among members of a design team responsible for different parts of the design. Each member of the design team is usually also a member of different research and engineering traditions which conceptualize problems differently and see the design as a whole, on the basis of different analogical models [125].

Another contention of the constructivist approach is that the consensus model aims at formal models of design. However, even within the sphere of formal models, a considerable degree of informal activities take place. Moreover, formal models are incomplete in their ability to trade off alternatives, which are mediated by inherently social processes. The connecting thread between these activities or approaches is that they are all expected to provide insight into the problem at hand. These activities are meant to facilitate a better understanding of needs and problems encountered, and potential solutions [126]. The resulting modeling activities can be modified into modeling activities which are results of both socio-linguistic, and more precise formal languages. Formal models are evaluated along several dimensions: accuracy, applicability, intent, and mutual consistency. Still, the very definition of these dimensions is a negotiated outcome of the design process, and not an input, a priori or otherwise [126]. The process of acceptance of formal methods is based on their reliability in a situated context of the domain of application [125, 126]. For example, the use of optimization methods in chemical engineering is much more accepted and stable than in mechanical engineering.

Several empirical findings and proposals (pertaining to how designers work) seem to support the social constructivist view [125, 126, 128]:

- Different designers and social groups use different vocabularies to describe the same or very closely related sets of things [129]. For example, for some, the artifact air tire introduced in the bicycle, was a solution to the vibration problem of small-wheeled vehicles. For others, the air tire was a way of going faster. For yet another group of engineers, it was an ugly looking way of making the low-wheeler even less safe (because of side-slipping) than it already was [127].
- Engineers typically spent a significant portion of their time (more than 50%) in documenting and communicating - much of it in the form of formal or informal negotiations. The negotiation most often takes the forms of one-on-one meetings and paper being passed about within the organization.
- The sociocultural, political, legal, and ecological situation of a social group shapes its norms and values, which in turn influence the meaning given to the artifact. Because different meanings can constitute different lines of

development, this seems to offer an operationalization of the relationship between the wider context and the actual content of technology.

- “Simultaneous” or “concurrent” engineering are terms that have gained a lot of currency recently. In each of these uses of the term, the underlying premise is that traditional design processes lack information on the later phases of the product realization process (such as production and operation) in the early phases of the development of the product [125]. In a study analyzing the traditional design and development process Danko and Prinz [130] conclude that successful design depends on the exchange of information between appropriate groups in the process. Further, rework and redesign are rampant, because field recognized conflicts often require design and field changes. Clark and Fujimoto [131] conclude, based on their study of American, Japanese, and European automobile companies, that Japanese firms are organized to maximize knowledge sharing between suppliers and the parent, and through very quick problem solving cycles that involve separate departments. This integration of problem solving helps reduce mistakes and rework and enables tool and die shops to handle changes with fewer transactions and less overhead [131].

In interpreting these findings, note that the underlying phenomenon being described is that of collaboration (or, in the traditional approach, lack thereof) among individual separated by disciplines or functional responsibility. However, effective simultaneous engineering is effective interdisciplinary design which, in turn, is the creation of effective *shared meaning*, the persistent form of it being *shared memory* [125]. Konda et al [125] define shared memory as the taking-hold of shared meaning created in specific design situations and applied to other design situations. Shared memory needs a substrate, or an infrastructure, in which the initial construction of a shared language are stored and perhaps reactivated at a later time on the same or a different project. Shared memory is distinguishable from shared meaning in that the former can have a more physical existence in, for example, databases, cross-indexes, models, papers, and so on [125, 126].

The phrase shared memory is used to denote the increasingly detailed aspect of a given profession's knowledge (*vertical shared memory*), as well as the sharing of meaning among multiple disciplines, groups, and group members (*horizontal shared memory*). Vertical shared memory is the codified corpus of knowledge, techniques, and models that exist in every professional group. The need for horizontal sharing among multiple disciplines arises from the general observation that engineering product development is a collaborative process, where engineers from diverse disciplines cooperate to specify, design, manufacture, test, market, and maintain a product [128]. The need for horizontal sharing *within* a profession arises from the differences in functioning contexts and meaning. For example, the beginning of the bicycle's development brought out two different kind of social groups within the profession of cyclists: (1) the social group of cyclists riding the high-wheeled Ordinary consisted of “young men of means and nerve” [132]. For this social group the function of the bicycle was primarily for sport; and (2) some parts of the safety low-wheelers and the safety ordinaries can be better explained by including a

separate social group of feminine cyclists.

Shared memory needs to be established in and nurtured by appropriate organizational structures incorporating the necessary technical substrates through which information, tools and, most importantly, people interact. Shared memory needs a substrate, or an infrastructure, in which the initial construction of a shared language are stored and perhaps reactivated at a later time on the same or a different project [125]. The substrate must address the following objectives:

- Facilitate capturing the context and history of a particular design (cases of both formal and informal modeling) in order that the experience (good or bad) can be reused within another design context;
- Facilitate the negotiation among designers and with other relevant parties (such as users);
- Forecast the impact of design decisions on manufacturing;
- Provide designers interactively with detailed manufacturing process planning;
- Develop mechanism for reaching agreements among designers through consensus standardization.

Following the general observation that the majority and most critical of activities in design are informal [126], no single representation or abstraction technique can be imposed on designers *a priori*, without severely limiting their capability to model. Therefore, the substrate of shared memory should allow multiple classifications, languages and methods designers find appropriate to carry out the above activities. Thus, the substrate can benefit from research on tools developed in Artificial Intelligence (AI), such as qualitative physics, semantic network representations, rule structures, machine learning, information retrieval techniques (relational databases), hypermedia, graph grammars, etc.

To achieve the objectives outlined above, several system architectures - based on current trends in programming methodologies, object-oriented databases, and knowledge based systems were developed. For example, the *n-dim* project, currently underway at the Engineering Design Research Center, Carnegie Mellon University, is a computer environment to support collaborative design [133]. *n-dim* is also a history capturing mechanism for complex corporate activities such as design. Other important aspects of *n-dim* are a task-level view for configuring and managing the design process, and an information-management system that allows for defining and displaying a user's current design context by means of *models* (linked information objects; see [126]). Other related work include the DICE (Distributed and Integrated environment for Computer-aided Engineering) being pursued at the Massachusetts Institute of Technology [128], the STEP/PDES effort, the RATAS project [134], the EDM model [135], and the spatial representation work being pursued at Carnegie Mellon University [136] are relevant tools that support collaboration among distributed teams of persons carrying out a complex process such as design.

Although automation facilitates the creation of effective shared memory, Clark and Fujimoto [131] argue that "competitive advantage will lie not in hardware and commercial software, but in the organizational capability to develop proprietary

software and coherently integrate software, hardware, and “humanware” into an effective systems.” For example, some degree of horizontal memory (which derives collaborative behavior) can be achieved based on *reward structures* to encourage the exchange of information among multiple disciplines, groups, and groups members. Hence, enhancing communication between human designers from different perspectives is feasible using organizational methods such as assignment of team responsibility and proximity of the designers, or through techniques such as Quality Function Deployment (QFD) for matching quality control and customer preferences [137]. In short, automation should not drive context - automation should be driven by context.

2.5 SCIENTIFIC STUDY OF DESIGN ACTIVITIES

The field of design theory is relatively new, which has been particularly stimulated by three computer-related technological advances: computer-aided design (CAD) [9], knowledge-based expert systems (KBES) [117], and concurrent engineering (CE) [100]. The main source of the slow development and confusion about design theory is that engineering design lacks the sufficient scientific foundations. Dixon [25] argued that engineering design education and practice lack an adequate base of scientific principles, and are guided too much by the specialized empiricism, intuition, and experience. Kuhn [58, 59] concluded that design is at a prescience phase and it must go through several phases before it constitutes a mature science (hence theory), which is that state of a discipline in which there is a coherent tradition of scientific research and practice, embodying law, theory, application and instrumentation. In order to achieve this kind of maturity, designers must borrow the methodologies from other disciplines (such as artificial intelligence, neural networks, logic and fuzzy logic, object oriented methods) that have reached relative scientific maturity [20].

A design method (within a design paradigm) does not constitute a theory; theory emerges when there is a testable explanation of why the method behaves as it does [25]. Design methods do not attempt to say what design is or how human designers do what they do, but rather provide tools by which designers can explain and perhaps even replicate certain aspects of design behaviors. The major components and aims of design theories are:

1. To construct a systematic inquiry into a phenomenon which is to uncover some intelligible structure or pattern underlying the phenomenon. That is, a theory of design must be in part descriptive [22];
2. Theories must be in part prescriptive, that is has the capability of specifying how design should be done, and allow us to construct more rational methods, and tools to support practical design;
3. Theories must be simple; that is, when two design theories are possible, we provisionally choose that which our minds adjudge to be the simpler, on the supposition that this is the more likely to lead in the direction of the truth. It

includes as a special case the principle of William of Occam - "Causes shall not be multiplied beyond necessity";

4. Theories must be consistent with whatever else we know or believe to be true about the universe in which the phenomenon is observed;
5. The value of a design theory be determined, to a great extent, by its generality and domain-independence (including mechanical engineering, electrical engineering, civil engineering and computer science).

While discussions of design theory, from a number of different perspective, have appeared in [105, 45, 49, 21, 52, 108, 35], in the following sections an attempt is made to demonstrate and concentrate on two interesting design theories - the axiomatic theory of design, and design as scientific problem-solving. The former theory is more grounded in the 'real' environment of design while the latter theory is a more abstract, speculative or philosophical.

2.5.1 THE AXIOMATIC THEORY OF DESIGN

The axiomatic theory of design is a structured approach to implementing a product's design from a set of functional requirements, that was developed by [113, 112]. It is a mathematical approach to design which differentiates the attributes of successful product and demonstrates design that are not manufacturable. Suh defines design as the culmination of synthesized solutions (in the form of product, software, processes or system) by the appropriate selection of design parameters that satisfy perceived needs through the mapping from functional requirements in the functional domain to design parameters in the structure domain. Suh pointed out that the fact that empirical decisions often lead to superior designs (as supported by technological progress) indicates that there exists a set of underlying principles, heuristics or axioms which govern the decision making process. If these axioms can be formalized and their corollaries derived, then it should be possible to establish a scientific basis for guidelines in manufacturing design [113]. The axiomatic approach is based on the following premises:

1. There exist a small number of axioms which will always lead to superior decisions in terms of increased overall productivity;
2. These heuristics may be established and examined through an empirical studies;
3. Axiomatic is not a mechanism for generating acceptable designs, but a tool to aid in the decision making process.

The hypothesized principles of manufacturing axiomatic may be given as two axioms:

1. *The Independence Axiom* - In an acceptable design, the design parameters and the functional requirements are related in such a way that specified design parameter can be adjusted to satisfy its corresponding functional requirement

without affecting other functional requirements (functional independence is analogous to the concept of orthogonality in linear algebra.) The independence axiom does not imply that a part has to be broken into two or more separate physical parts, or that a new element has to be added to the existing design. Functional decoupling may be achieved without physical separation, although in some cases such physical separation may be the best way of solving the problem (recall the bicycle's evolution);

2. *The Information Axiom* - The best design has minimum information content. It coincides with the principle of 'simplicity'. The search for simplicity, likewise the search for beauty, is a powerful aesthetic imperative that serves as a basic component of a designer's value system. Simple in the design means, for example, being able to minimize the number and complexity of part surfaces. The simplicity principle implies that if a design satisfies more than the minimum number and measure of functional requirements originally imposed, the part or process may be overdesigned. Simple design will result in reducing product cost (such as inventory and purchasing costs), and enhancing quality.

Suh et al. have developed a number of theorems and corollaries which may readily be shown to be implied by the axioms. These corollaries bear strong resemblance to many design guidelines and design for manufacture (DFM) techniques (recall the ASE design paradigm) that were developed in manufacturing companies [12]. Some of these guidelines (corollaries) include: minimizing functional requirements, decoupling of coupled design, integration of physical parts, symmetry, standardization and largest tolerance.

2.5.2 DESIGN AS SCIENTIFIC PROBLEM-SOLVING

Discussions in the literature of design processes generally treats separately a category of the artificial sciences (engineering disciplines) and the natural sciences (such as physics, biology and geology). Several criteria that demarcate the artificial sciences from the natural sciences have been identified [105, as well as others]:

1. Engineers are concerned with how things ought to be, that is in order to attain goals, and to function while science concerns itself solely with how things are. In an ultimate philosophical sense, the natural science has found a way to exclude the normative, and to concern itself with solely with descriptive aspects of nature;
2. Engineering is concerned with 'synthesis' while science is concerned with 'analysis';
3. Engineering is 'creative, intuitive and spontaneous' while science is 'rational and analytic'.

Schön [98] has proposed that design creates an entirely new epistemology (epistemology is concerned with the question of what knowledge is and how it is

possible), which he terms ‘reflection-in-action’ as contrasted to scientific discovery which concerns with ‘technical rationality’. Cross [21] and Coyne et al. [20] have claimed that the aims of design and those of science differ. Coyne et al. summarized elegantly the demarcation between the natural science and design as: “science attempts to formulate knowledge by deriving relationships between observed phenomena. Design, on the other hand, begins with intentions and uses the available knowledge to arrive at an entity possessing attributes that will meet the original intentions. The role of design is to produce form or more correctly, a description of form using knowledge to transform a formless description into a pragmatic discipline concerned with providing a solution within the capacity of the knowledge available to the designer. This design may not be ‘correct’ or ‘ideal’ and may represent a compromise, but it will meet the given intentions to some degree”.

That there are indeed differences in aims will be agreed in general. Unfortunately, this demarcation of the natural sciences from engineering as a result of the differences in their respective aims has led to the fictitious attitude that the methodology of science and engineering are fundamentally different. Laudan [66] stresses at the outset of his essay *Progress and its Problems* that scientific problems are not fundamentally different from other kinds of problems, though they are different in degree. Indeed, we shall show that this view can be applied, with only a few qualifications, to all intellectual disciplines, and design activity in particular. Engineers wishing to construct an artifact capable of implementing a process (such as problem solving) often study naturally occurring systems that already implement the process. This approach has led, inter alia, many researchers to investigate psychological models of human thought as a basis for constructing the constituent methods within the AI design paradigm [79, 119]. We shall describe in the sequel those theories (which conclude, among others, [105, 55, 28, 5, 1, 22]) that support the thesis that the engineering and natural science are methodologically indistinguishable, and that the structure of problem solving of scientific communities (scientific discovery) can be used to justify many of the decisions encountered in the design process. The relationship is one that can be seen at a high level of abstraction. The most important aspects of this relationship are forms of scientific discovery which pursue the hypothetico-deductive (H-D) or the related procedure of abductive inference, and Kuhn’s model of scientific progress [58, 59, see also Section 2.4.1] whose primary element is the “paradigm” (and more elaborated ideas such as the notion of research programmes by Lakatos [61, 62], and Laudan’s [66] idea of research tradition). We focus our attention on scientific progress and how scientific communities solve problems, following by a brief outline of the parallelism to design processes. We hope to gain useful insight from these parallelism that will aid us to corroborate the thesis that design and natural science are methodologically indistinguishable.

The publication of Thomas Kuhn’s *The Structure of Scientific Revolutions* in 1962 was an important milestone in the development of the historiography of science. It was the first attempt to construct a generalized picture of the process by which a science is born and undergoes change and development envisaged by Kuhn’s model and further developed and refined by [61, 66]. Their approach may be summarized as

follows (see Figure 2.10):

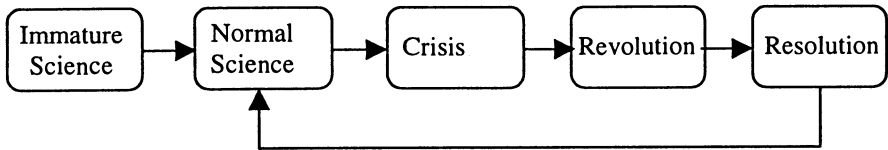


Figure 2.10 Five Stages in the History of Scientific Disciplines According to Kuhn

1. *Immature Science*: A pre-paradigm stage in which the natural phenomena that later form the subject matter of a mature science are studied and explained from widely differing points of view [58, 59].

2. *Normal Science*: The emergence of a paradigm (e.g., Newtonian mechanics, quantum mechanics), embodied in the published works of one or more great scientists, defining and exemplifying the concepts and methods of research appropriate to the study of a certain class of natural phenomena, and serving as an inspiration to further research by its promise of success in explaining those phenomena.

A period of normal science conducted within a conceptual and methodological framework derived from the paradigmatic achievement, involving actualization of the promise of success, further articulation of the paradigm, exploration of the possibilities within the paradigm, use of existing theory (a set of hypotheses) to predict facts, solving of scientific puzzles, development of new applications of theory, and the like. Lakatos, as well as Laudan, contend that science is seldom dominated by just one paradigm, as Kuhn claims in his account of normal science, but rather that competition between paradigms generally co-occurs with processes of development within a paradigm. Lakatos replaces Kuhn's term paradigm with the term research programme (for example, we can distinguish between the flat-world and round-world research programmes). The common thread linking different theories into a common research programme is a "hard core" of basic assumptions shared by all investigators. This core is surrounded by a "protective belt" of auxiliary assumptions. The "hard core" which may consist of assumptions such as "No action at a distance," remains intact as long as the research programme continues, but researchers can change the auxiliary assumptions in the protective belt to accommodate evidence that either has accumulated or is developed in the course of research. Laudan invokes the idea of a large-scale unit in science that he calls a "research tradition". Like Lakatos' research programmes, research traditions for Laudan consist of a sequence of theories, but they lack a common core that is immune to revision. What holds a research tradition together are simply common ontological assumptions about the nature of the world and methodological principles about how to revise theories and develop new theories.

Kuhn, Lakatos and Laudan agree that the main activity of scientists (within a paradigm, research programme or tradition) is problem solving. Scientific problem

solving consists of generating hypotheses to account for phenomena, and procedures for substantiating and refuting these hypotheses. The Positivists called the former procedure (substantiation) for developing scientific theories the hypothetico-deductive (H-D) method. Popper has called the latter procedure (refutation) conjecture and refutation (C-R). The basic idea of the H-D method is that scientists begin with a phenomena (anomaly, experimental or conceptual problem) that requires explanation. Having developed a hypotheses (for the Positivists how hypotheses were arrived at was not a matter for logical inquiry), the task was to test and discover whether the hypotheses was true. If it was; it could provide the theory needed to explain the phenomena. The hypothesis is a general statement and so could be tested by considering initial hypothesis, and deriving predictions about what would happen under these hypothesis. If these predictions turn out to be true, the initial hypothesis would be confirmed; if the predictions turn out false, the hypothesis would be disconfirmed. In either case, a new problem (phenomena) would have been generated and the cycle begins once more. The Positivists thought that at least positive tests of a hypothesis could give support to that hypothesis. Popper [86] contended that this assumption was false; observation statements (“this swan is white”) can never logically imply theories (“all swans are white”) but they can logically refute them (by providing a counterexample of just one black swan). He proposed instead that scientists should begin by making conjectures about how the world is and then seek to disprove them. If the hypothesis is disproved, then it should be discarded. If, on the other hand, a scientist tries diligently to disprove a hypothesis, and fails, the hypothesis gains a tentative or conjectural stature. Although failure to disprove does not amount to confirmation of the hypothesis and does not show that it is true or even likely to be true, Popper speaks of such an hypothesis as corroborated. The virtue of a corroborated hypothesis is that it is at least a candidate for being a true theory, whereas hypotheses that have been disproved are not even candidates. Popper terms this the process of conjectures and refutations (C-R).

3. *Crisis*: A crisis stage of varying duration precipitated by the discovery of natural phenomena that “violate the paradigm-induced (research programme or tradition) expectations that govern normal science” and marked by the invention of new theories (still within the prevailing paradigm) designed to take account of the anomalous facts.

4. *Revolution and Resolution*: A relatively abrupt transition to a new paradigm (research programme or research tradition) that defines and exemplifies a new conceptual and methodological framework incommensurable with the old; and continuation of normal science within the new paradigm (research programme or tradition).

In addition to the foregoing properties, let us state the following important features:

- Conjectures often develop by adjusting to new problems by discovering the

discrepancy between their predecessors and the phenomena (recall Newell and Simons' General Problem Solver scheme). This process led, for example, to better understanding of Euler's formula and the concept of "polyhedron" [see 62];

- When an anomaly is discovered in a theory, the outcome is often an adjustment [62] of the theory rather than a total dismissal of the existing paradigm (which defines the character and structure of the original theory), research programme or tradition. Most theories evolve through a continual and incremental activity (recall the concept of 'incrementalism' given by [69] in the context of design);
- Although corroboration does not give us a logical basis for increasing our confidence that the hypothesis will not be falsified on the next test, it does serve to limit us to an ever narrowing set of hypotheses that might be true. Popper has compared this process to Darwinian natural selection, for both nonadapted organisms and false theories are weeded out, leaving the stronger to continue in the competition. Evolution is often gradual, taking years or decades. Other theorists have pursued the idea that theory development may be parallel to the process of evolution by natural selection [see, for example, 58, 59, 16, 118];
- As pointed out above, for the Positivists how hypotheses were arrived at was not a matter for logical inquiry, since discovery was assumed to be nonrational process. Hanson [40] was one of the first to urge philosophers to redirect attention to discovery (extending the context of 'scientific justification'). His proposal was to pursue what the 19th century American Pragmatist, Charles Peirce, called abductive inference which is similar to the H-D method. The alternative to deductive reasoning is generally taken to be induction. One of the things that has brought about renewed interest in discovery is the recognition, partly motivated by work in empirical psychology, that human reasoning involves additional modes of reasoning than deductive logic and enumerative induction such as "common sense" knowledge, gestalt-like perception, analogical reasoning or by sheer trial and error. Because scientific reasoning is simply an extension of ordinary human reasoning, there is reason to think that such strategies figure also in science. Newell and Simon [79] popularized the idea that in solving complex problems we rely on heuristic principles that simplify the process through which we search for a solution. Recently there has been considerable interest by both philosophers and those in Artificial Intelligence in using AI as a tool for studying scientific reasoning [64];

Following our previous discussions on descriptive properties of design, especially the adaptive and evolutionary properties discussed in Section 2.3.2, and the prescriptive role of design paradigms, it is plausible to believe the hypothesis that there is a direct and striking resemblance between the structure of design processes and the foregoing structure of problem solving of scientific communities is corroborated (following Popper). Figure 2.11 illustrates the parallel relationship between inquiry that is associated with science and that which is associated with engineering design. This correspondence is summarized as follows:

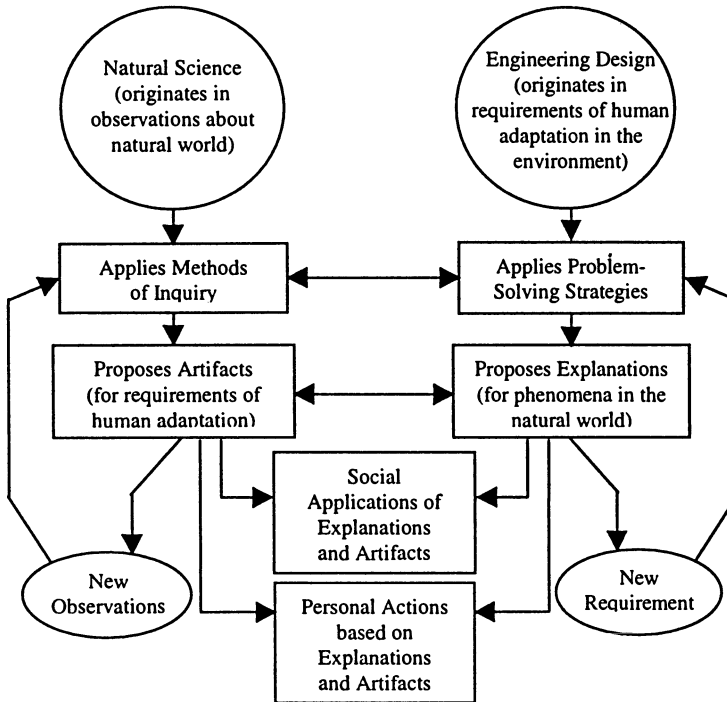


Figure 2.11 The Interrelationships between the Science and Engineering Design

- The counterpart of the Kuhnian paradigm or Laudan's research tradition is the designer's knowledge-base needed to generate the set of design solutions. The designer's knowledge-base is a coherent tradition of knowledge and practice, embodying theories (such as stress analysis, thermodynamics analysis, information on materials and the like), design paradigms, applications, heuristics (transformation operators), instrumentation, and law (such as existing zoning regulations). The most versatile designers will represent much of their knowledge about their environments declaratively. The set of knowledge underlies the designer's conceptual assumptions and world view and values serving as best approximation to what the design domain actually is; so, we really should be talking about the designer's beliefs rather than the designer's knowledge. But, following the tradition established by the phrase "knowledge-based systems", we will speak of the designer's knowledge;
- The counterpart of a set of phenomena, events or problems are design problems that are entirely characterized by and generated as a result of measurable and non-measurable requirements. Laudan distinguishes two kinds of problems that can confront a research tradition - empirical inadequacies of the current theories and conceptual problems with the theories comprising the tradition. It is with a direct correspondence to the distinction (see Section 2.2.4) between empirical, measurable or well-defined requirements (which specifies externally observable

- or empirically determinable qualities for an artifact); and ill-defined requirements (conceptual);
- The counterpart of a scientific theory (set of hypotheses) is the tentative design/form (constitutes a systematic representation of functional relationships of the components; see II-*H*) serving (much the same as scientific theories) as a vehicle for the designer to capture her thoughts, as a plan for implementation, and as a vehicle for reflecting the evolutionary history that led to the emergence of the final form/design, thus facilitating the inspection, analysis and redesign (change) of the artifact;
 - Scientific discovery follows the hypothetico-deductive (H-D) method, or the more justifiable procedure (following Popper) of conjecture and refutation (C-R). Moreover, Popper [as well as 58, 59, 16, 118] has compared the process of scientific discovery to Darwinian natural selection. It is with a direct correspondence with the evolutionary nature of design processes (see Section 2.3.2): as the design process develops (due to bounded rationality), the designer test (to corroborate or disprove the hypotheses) the tentative design against the requirements. If the test fails, the designer modifies either the tentative design or requirements (the counterpart of theory adjustment), so as to remove the discrepancy (recall how Euler's formula was adjusted to a new "counterexample"; see [62]) between them, and converge or establish a fit between the two parts. Testing involves a wide range of reasoning types, such as classical and approximate logical systems (theorem provers, qualitative reasoning); experiments (e.g., simulations); knowledge accumulated from previous experience; and common sense reasoning (heuristics and 'rules of thumb'). If a problem is found as the result of testing, it also becomes a new problem to be solved in another design cycle;
 - Besides the embodiment of ontogenetic design evolution within the model of scientific discovery, Kuhn's model of scientific progress can also explain phylogenetic design evolution (see Section 2.3.2). Incremental redesign activity corresponds to the continual and incremental evolvement of scientific theories within a normal science, whereas innovative redesign activity corresponds to a transition to a new paradigm (conceptual or paradigm-shift);
 - Comparison of theories in terms of "degree of falsifiability", a concept most fully developed by Popper [86] and Newtonian world-view of "reductionism", cast considerable light on the axiomatic theory of design [113]. Popper concludes that the best reason for entertaining a theory is that it is testable (more accurately "falsifiable") - i.e., that it makes strong predictions that are readily capable of being refuted by evidence if the theory is, in fact, false. Let us, following Popper, restate the criterion in a number of other forms. Theory T_1 (e.g., "All heavenly bodies move in circles") is decidedly stronger than Theory T_2 ("All planets move in ellipses") if it is more universal and precise. Since in a theory it is desirable to obtain the maximum of prediction from the minimum of assumptions, the more universal and precise a theory, hence the more falsifiable, the better. In the case of two theories related as T_1 and T_2 we will entertain the weaker, T_2 , only if the stronger, T_1 , is falsified. Another form is that of

simplicity. “Simple” theories are generally thought preferable to “complex” theories. Consequently, for the same design problem, a computer time-sharing system design, T_1 , which was accompanied by larger amounts of experimental work is stronger than a computer time-sharing system accompanied by poor amounts of experimental work. Similarly, as previously noted, Suh’s information axiom and derivations (such as the minimization theorem and corollaries, see [113]) are derived from Popper’s (and Occam’s razor) simplicity principle.

What we propose to do now is to briefly indicate how Suh’s independence axiom and derivations, as well as various hierarchical decomposition principles of design problems represent a direct embodiment (from its paradigmatic aspect) of the 17th-century Newtonian mechanics. The essential point of the Newtonian reductionistic language is that, in the Newtonian picture, the categories of causation are isolated into independent mathematical elements of the total dynamics. Indeed, the independence axiom and the principle of a nearly decomposable system (A hierarchical system of components C_1, \dots, C_n where each C_i is itself an aggregate of more primitive entities such that the interactions between the entities within the C_i ’s are appreciably stronger than those between the C_i ’s is called a nearly decomposable system, see [105]) are nothing but a paraphrase of the Newtonian language, adapted to inherently non-mechanical situation.

- Both designer’s knowledge-base and Laudan’s research tradition have the component of a group’s shared past cases (used within the case-based design paradigm) and examples. By that it means the concrete problem-solutions and cases that students encounter from the start of their scientific and engineering education, whether in laboratories, on examinations, or at the technical problem-solutions found in the periodical literature and information about wide variety of devices (their parts, characteristics, materials, uses and behavior) found from commercial catalogs.
- The ASE design paradigm is based on a phase of thorough analysis of the requirements, followed by the actual synthesis of design, and then a phase of testing the design against the requirements, which is strikingly resemble to the one of the most influential methodologies of science, namely, inductivism. According to inductivism only those propositions can be accepted into the body of science which either describe hard facts by making observations and gathering the data (the counterpart of an analysis phase of requirements) or are infallible inductive generalizations (the counterpart of a synthesis phase of design) from them. The problem with inductivism (as well as with the ASE paradigm) is that the inductivist cannot offer a rational ‘internal’ explanation for why certain facts rather than others were selected in the first instance. Hanson [41] argued that observation itself is theory-laden, that is that what we perceive is influenced by what we know, believe, or are familiar with (which implies that designers have some conceptual models prior to gathering requirements).

The above discussion on the interrelationship between the science and

engineering design emphasizes *individual* human design problem solving. To summarize, Popper argues that theories ought to prove their mettle through several critical tests (theories should be falsified rather than confirmed). On the other hand, the approach of Kuhn, Laudan and Lakatos is historical and case-based. They argue that theories are always more than a single hypothesis. Theories are usually an interweaving of a number of law-like rule-like statements supplemented by an often larger number of auxiliary hypotheses concerning instruments used in testing them and specifications of initial conditions and experimental set-up necessary for these tests. Our analysis of design methodologies shows that these two alternative view of scientific development (logical reconstructability versus case-based approach) were utilized to gain useful insight on the parallelism between the methodologies of science and design methodology (emphasizing individual problem solving).

Thus far, we have not discussed the fact that design processes as well as scientific progress involve social factors (see Section 2.4.9 about a discussion on design as a social process); have similar mechanisms of organizing collaborative activities among multiple disciplines, groups, and groups members (e.g. negotiations); and have similar mechanism for allocating effort (such as funding structure and peer review). Indeed, the views of Kuhn, Laudan and Lakatos have been elaborated more recently by others who pointed out that social and sometimes cultural factors can be significantly involved in bringing scientific debate to a relative close and stabilizing concepts of facts [127, 138, 139, 140, 141]. These approaches have generated a vigorous program of empirical research on the process of the social construction of scientific knowledge in various sciences. See, for example, Collins [141] for scientific controversy, Collins and Pinch [142] for physics and biology, and DeMillo et al. [143] for program verification in computer science. These approaches, which we refer to as "*social constructivist*," mark an important new development in the *sociology of science*. The main characteristics of the social constructivist view is described by the "Empirical Programme of Relativism" (EPOR) as was developed in the sociology of scientific knowledge. Three stages in the explanatory aims of the EPOR can be identified [127]:

1. In the first stage (Interpretative Flexibility), it is shown that scientific findings are open to more than one interpretation. Often opposing directions of research are found between schools of thought that do not closely interact. For example, the Freudian and behaviorist schools in psychology explain human neuroses in very different ways. Neither has proved that its ideas are in all respects superior to its competitors. There is no good reason for the community as a whole to accept one theory over the other. In the absence of criterial experiments to discredit one theory or the other, these opposing ideas may coexist for a long time. Moreover, all thought happens in the context of certain assumptions. For example, we can distinguish between the *flat world* and the *round world* hypotheses by imagining two viewpoints, one for each of the theories. We would expect statements in the flat-world viewpoint to the effect that the world has edges and the belief in the round-world viewpoint that traveling west long enough will bring you back to where you started. Research effort in opposing

directions is not only found between distinct schools of thought; it can often be found in the same research group or individual.

2. When an anomaly is discarded in a theory, the outcome is often an adjustment of the theory rather than a total dismissal. This is because theories that have shown some success tend to develop a core of fundamental concepts that serve to motivate further work. The process of adjustment, then, is an effort to protect this core from being discredited. Evolution is often gradual, taking years or decades. On rare occasions it is quite rapid as the community reacts to a “breakthrough” that clearly decides an issue (controversy) previously muddled. Social mechanism that limit interpretative flexibility, and thus allow scientific controversies to be terminated are described in the *second stage* of EPOR.
3. The sociocultural and political situation of a social group shapes its norms and values, which in turn influence the meaning given to theories. A *third stage* is to relate the “closure mechanism” to the wider social-cultural milieu.

Having described the EPOR approach to the study of science, and the social constructivist approach to the study of design (as described in Section 2.4.9), we now discuss the parallels between them [127]:

1. *Interpretative Flexibility* - In design, there is flexibility in how designers interpret artifacts, and how artifacts are designed. There is not just one possible way or one best way of designing an artifact. For example, for some, the artifact air tire introduced in the bicycle, was a solution to the vibration problem of small-wheeled vehicles. For others, the air tire was a way of going faster. For yet another group of engineers, it was an ugly looking way of making the low-wheeler even less safe (because of side-slipping) than it already was [127]. Moreover, other artifacts were seen as providing a solution for the vibration problem, e.g. the saddle, the steering bar, and solutions used spring construction in the frame;
2. *Closure and Stabilization* - Closure in design involves the stabilization of an artifact and the resolving of technological controversy. Pinch and Bijker [127] identify two closure mechanisms. Rhetorical closure emerges when the relevant social groups *see* the design problem as being solved (although it might not be solved in the common sense of the word). For example, for some time, the “safety controversy” around the high-wheeler bicycles was resolved by announcing (through advertisement) that the artifact (i.e. the air tire) was perfectly safe although to engineers high-wheeled bicycles were known to have safety problems. Closure by redefinition of the problem means translating the meaning of the artifact to constitute a solution to quite another problem. For example, originally the air tire meant a solution to the vibration problem to the users of the low-wheeled bicycle. However, the group of sporting cyclists riding their high-wheelers did not accept that as a problem at all. Eventually, closure has been reached when the meaning of the air tire was translated to constitute a solution to quite another problem: the problem of how to go as fast as possible [127].

To summarize, we argue that the social constructivist view that is prevalent within the sociology of science provides useful insights that will aid in the interpretation of design processes with respect to the social factors involved.

2.6 A GENERAL DESIGN METHODOLOGY

This section presents a design methodology, based on the scientific community metaphor, by emphasizing the variational (or parametric) design part. In variational design, the dimensions of a part are calculated by solving a system of constraints or specifications (typically, nonlinear equations). Let us summarize some of the very basic features of the evolutionary design model as articulated in Chapter 6 and Chapter 17:

- In variational design, an artifact at any particular abstraction level is described in terms of part types (a group of objects which are similar but have different sizes). Every part can be described by a set of attributes. Each attribute can be described by its dimension (such as wire diameter, spring diameter, number of active coils, and modulus of elasticity).
- Specifications or constraints, at any particular abstraction level, are the various functional, behavioral, performance, reliability, aesthetic, or other characteristics or features that are to be present in the physically implemented artifact. In the case of the Gear Box design (see Chapter 17), the initial specifications were to design “a mass production device, which can be used for lifting light objects or opening a garage door in a family or small warehouse.” In variational design, closed-form constraints are usually either *Euclidean* (including distance, tangency, parallelism, and so on), or *functional* (such as mass properties, forces, stiffness, strength, rating life, and so on). A higher order constraint is a property that is satisfied by lower order constraints.
- Design proceeds as a succession of cycles. In each cycle, the objects that evolve are the design/specifications complexes; the evolution of the design/specifications complexes is towards the satisfaction of tentative specifications; and the *mechanism* employed in this evolutionary process is the attempt to verify the validity (“degree of believability”) of existing specifications. As a consequence of this mechanism, new specifications and design parameters are introduced. If the design satisfies the specifications, then there is a fit between the two, otherwise there is said to be a misfit between design and specifications. For example, the Gear Box design process is terminated when the transmission parts, casing, shaft set, and accessories are fully specified, and all user-specified requirements (duration, capacity), strength constraints and heat balance are satisfied by the current solution. This evolutionary design model follows the hypothetico-deductive (H-D) method, or the more justifiable procedure (following Popper) of conjecture and refutation (C-R) of scientific discovery (see Section 2.5.2).

The validity of specifications (*qualitative* design specifications as well as *closed-form equations*) as encountered by the designer is determined by means of (1) relevant *knowledge* of the design domain such as *tools* and *techniques* of verification (e.g. finite-elements or finite-differences); and (2) the process *history*, which includes both the sequence of all process states (i.e., the design/specifications complexes) visited so far, together with the transformations and production rules (or inference rules) used to modify such process states. Consider the following examples:

1. Constraint C_1 ("to insure a long work life for the rope") is considered validated only if constraint C_2 ("the overbending of the rope is avoided") is validated. This in turn, is considered validated only if constraint C_3 ("the minimum drum diameter should be large enough") is validated. According to the engineering recommendation value (knowledge of the design domain), for the minimum drum diameter to be large enough, a 15 to 20 times the rope diameter should be chosen. Therefore, the designer chooses the drum diameter as $d_{dr} = 3.75$ inches (a design parameter). The *tool* used for support of constraint C_3 is simply to check if the selected drum diameter is indeed 15 to 20 times the rope diameter;
2. The constraint "select an efficient reducer", is determined by a concept selection process (the *verification tool*), which is used to measure the degree of efficiency of each alternative with respect to well-defined criteria. This in turn, is considered validated only if the design parameter ("the reducer is a worm and wormgear type") is validated (or selected). The specifications and design parameters have dependencies among them. Dependencies between the specifications and parameters are represented by *rules*, or logical relationship the validity of which implies the validity of the specification or the design parameter. The rules are specified in the first-order calculus. For example, the validity of specification C ("the shear strength and compression stress of the key must be less than the allowable shear strength and compression stress, respectively") is determined by the logical relationship: ("the shear stress should be calculated" \wedge "the compression stress should be calculated" \wedge "the key's material should be specified" \wedge "the shear stress must be less than the allowable shear stress of the material" \wedge "the compression stress must be less than the allowable compression stress of the material"). In this case, the specification part is updated by replacing specification C by its *antecedents* (C is called *consequent*, see [149]). The specification "the key's material should be specified" derives a new design parameter DP "the material of the key is ASTM 40." In this case, the *design part* is updated by adding the new design parameter DP.

A most distinct feature of the design evolutionary model is that the design process is *constantly subject to revision*. For example, in the Gear Box design process, if the shear strength and compression stress specifications are not satisfied by the key, the designer has either to change the key material (DP₁), or change the shaft diameter DP₂ (the key size is associated with the diameter of the shaft). In that

case, all specifications that dependent on DP_1 and/or DP_2 will have to be *revised* in the light of this new design state. The evolutionary design process is, thus, *non-monotonic* in nature in accordance with Kuhn's model of scientific progress (see Section 2.5.2).

2.7 SUMMARY

To conclude this chapter it is useful to restate that design contains a wide range of concepts. Design begins with the acknowledgment of needs and dissatisfaction with the current state of affairs and realization that some action must take place in order to solve the problem. Design science is a collection of many different logically connected knowledge and disciplines constituting miscellaneous design paradigms. Although there is no single paradigm that can provide a complete definition of the design process, there are common characteristics that form the framework within which various paradigms are utilized.

We maintained that the demarcation of the natural sciences from engineering design, as a result of the differences in their respective aims, has led to the fictitious attitude that the methodologies of science and engineering design are fundamentally different. Alternatively, we display the parallelism between the natural sciences and engineering design. In summary we believe that the scientific community metaphor (as captured by a number of philosophers, sociologists and historians of science) can supply important insights that will aid in the interpretation (a descriptive role) and construction (a normative role) of design problem solving systems. The resulting framework has been useful in guiding the development of general purpose design process meta-tools as shown in Parts III and IV of the book. Finally, we may realize that design theorists can expect no easier fate than that which befell scientists in other disciplines.

REFERENCES

1. Agassi, J., *Technology: Philosophical and Social Aspects*. Tel-Aviv: Open University, 1985.
2. Akin, O., "An Exploration of the Design Process," *Design Methods and Theory*, Vol. 13 (3-4), pp. 115-119, 1979.
3. Alagic, S. and Arbib, M.A., *The Design of Well-Structured and Correct Programs*. Berlin: Springer-Verlag, 1978.
4. Alexander, C., *Notes on the Synthesis of the Form*. Cambridge MA: Harvard University Press, 1964.
5. Altshuller, G.S., *Creativity as an Exact Science*. New York: Gordon and Breach Publishers, 1984.
6. Antonsson, E.K., "Development and Testing of Hypotheses in Engineering Design Research," *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 109, pp. 153-154, 1987
7. Asimow, W., In *Emerging Methods in Environment Design and Planning* (ed. Moore G.T.). Cambridge. MA: MIT Press, pp. 285-307, 1962.
8. Bell, C.G. and Newell, A., *Computer Structures: Reading and Examples*. New York: McGraw-Hill, 1971.
9. Bezier, P.E., "CAD/CAM: Past, Requirements, Trends," In *Proc. CAD*, Brighton, pp. 1-11, 1984.
10. Billington, D.P., *Robert Maillart's Bridges: The Art of Engineering*. Princeton. NJ: Princeton University Press, 1979.

11. Bobrow, D.G., *Qualitative Reasoning About Physical Systems: An Introduction.* Cambridge, MA: MIT Press, pp. 1-5, 1985.
12. Boothroyd G. and Dewhurst P., *Product Design for Assembly.* Wakefield, RI: Boothroyd & Dewhurst Inc, 1987.
13. Braha, D. and Maimon, O., "A Mathematical Theory of Design: Modeling the Design Process (Part II)," *International Journal of General Systems*, Vol. 25 (3), 1997.
14. Brown, D.C. and Chandrasekaran, B., "Expert Systems for a Class of Mechanical Design Activity." In [33], pp. 259-282, 1985.
15. Brown, D.C. and Chandrasekaran, B., "Knowledge and Control for a Mechanical Design Expert System," *Computer*, Vol. 19 (7), July, pp. 92-100, 1986.
16. Campbell, D.T., "Evolutionary Epistemology." In *The Philosophy of Karl Popper*, P. Schlipp (ed.), LaSalle, IL: Open Court, 1974.
17. Chandrasekaran, B., "Design Problem Solving: A Task Analysis," *AI Magazine*, Winter, 1990.
18. Charniak, E. and McDermott, D., *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley, 1985.
19. Churchman, C. W., "The Philosophy of Design," *ICPDT*, Boston, Mass, August., pp. 17-20, 1987.
20. Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M. and Gero, J.S., *Knowledge-Based Design Systems.* Reading, MA: Addison-Wesley, 1990.
21. Cross, N. (ed.), *Development in Design Methodology.* New York: John Wiley, 1984.
22. Dasgupta, S., "The Structure of Design Processes," In *Advances in Computers*, Vol. 28, M.C. Yovits (ed.). New York: Academic Press, pp. 1-67, 1989.
23. Diaz, A. R., "A Strategy for Optimal Design of Hierarchical System Using Fuzzy Sets," In *NSF Engineering Design Research Conference*, University of MASS., Amherst June, pp. 11-14, 1989.
24. Dieter, G.E., *Engineering Design: A Materials and Processing Approach.* New York: McGraw-Hill, 1983.
25. Dixon, J.R., *AI EDAM*, Vol. 1 (3), pp. 145-157, 1987.
26. Dong, Z., "Evaluating Design Alternatives and Fuzzy Operations," *International Conference on Engineering Design*, Boston, MASS, August, pp. 17-20, pp. 322-329, 1987.
27. Fennes, S.J., Flemming, U., Hendrickson, C., Mehar, M.L. and Schmitt, G., "Integrated Software Environment for Building Design and Construction," *Computer Aided Design*, Vol. 22 (1), pp. 27-35, 1990.
28. Fetzer, J.H., "Program Verification: The Very Idea," *Comm. ACM*, Vol. 31 (9), September, pp. 1048-1063, 1988.
29. *FMS Handbook*, CSDL-R-1599 U.S Army Tank Automotive Command under contract No. DAAE07-82-C-4040, 1983.
30. Freeman, P. and Newell, A., "A Model for Functional Reasoning in Design," In *Proc. of the 2nd Int. Joint Conf. on Artificial Intelligence*, pp. 621-633. 1971.
31. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A guide to the Theory of NP-Completeness.* San Francisco: W. H. Freeman and Company, 1979.
32. Gero, J.S., "Prototypes: A New Schema for Knowledge Based Design," *Technical Report*, Architectural Computing Unit, Department of Architectural Science, 1987.
33. Gero, J.S. (ed.), *Knowledge Engineering in Computer-Aided Design.* Amsterdam: North-Holland, 1985.
34. Gero, J.S. and Coyne, R.D., "Knowledge-Based Planning as a Design Paradigm," In *Design Theory for CAD*, H. Yoshikawa and E.A. Warman (eds.). Amsterdam: Elsevier Science Publishers, 1987.
35. Giloi, W.K. and Shriver, B.D. (eds.), *Methodologies for Computer Systems Design.* Amsterdam: North-Holland, 1985.
36. Glegg, G.L., *The Science of Design.* Cambridge, England: Cambridge University Press, 1973.
37. Goel, V. and Pirolli, P., "Motivating the Notion of Generic Design with Information-Processing Theory: The Design Problem Space," *AI Magazine*, Vol. 10 (1), pp. 18-38, 1989.
38. Gould, S. J., *Ontogeny and Phylogeny.* Cambridge, MASS: Belknap Press of the Harvard University Press, 1977.
39. Gregory, S.A., "The boundaries and Internals of Expert Systems in Engineering Design," *Proceeding of the Second IFIP Workshop on Intelligent CAD*, pp. 7-9, 1988.
40. Hanson, N.R., "An Anatomy of Discovery," *The Journal of Philosophy*, Vol. 64, pp. 321-352, 1967.
41. Hanson, N.R., *Patterns of Discovery.* Cambridge, England: Cambridge University Press, 1958.

42. Harrisberger, L., *Engineersmanship*. Belmont, California: Brooks/Cole Publishing, 1966.
43. Henderson, P., *Functional Programming: Application and Implementation*. Englewood Cliffs, NJ: Prentice-Hall International, 1980.
44. Holton, G., *Introduction to Concepts and Theories in Physical Science*. Reading, MA: Addison-Wesley, 1952.
45. Hubka, V., *Principles of Engineering Design*. London: Butterworth Scientific, 1982.
46. Hubka, V. and Eder, W.E., *Theory of Technical Systems: A Total Concept Theory For Engineering Design*. Berlin: Springer-Verlag, 1988.
47. Huhns, M. H. and Acosta, R. D., "Argo: An Analogical Reasoning System for Solving Design Problems," *Technical Report AI/CAD-092-87*, Microelectronic and Computer Technology Corporation, March, 1987.
48. Ishida, T., Minowa, H. and Nakajima, N., "Detection of Unanticipated Functions of Machines," In *Proc. of the Int. Symp. of Design and Synthesis*, Tokyo, pp. 21-26. 1987.
49. Jaques, R. and Powell, J.A. (eds.), *Design: Science: Method*. Guildford, England: Westbury House, 1980.
50. Johnson-Laird, P.N., *The Computer and the Mind*. Cambridge, MA: Harvard University Press, 1988.
51. Jones, J.C. 1963, "A Method of Systematic Design," In *Conference on Design Methods* (J.C. Jones and D. Thornley, eds.), pp. 10-31, Pergamon, Oxford. Reprinted in [21].
52. Jones, J.C., *Design Methods: Seeds of Human Futures* (2nd Edition). New York: John Wiley, 1980.
53. Kant, E. and Newell, A., "Problem Solving Techniques for the Design of Algorithms," *Information Processing and Management*, Vol. 20 (1-2), pp. 97-118, 1984.
54. Kolodner, J. L., Simpson, R. L. and Sycara, K., "A Process Model of Case-Based Reasoning in Problem Solving," *Proceedings of IJCAI-85*, Los Angeles, pp. 284-290, 1985.
55. Kornfeld, A.W. and Hewitt, E.C., "The Scientific Community Metaphor," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11 (1), 1981.
56. Krishnamoorthy, C.S., Shivakumar, H., Rajeev S. and Suresh, S., "A Knowledge-Based Systems with Generic Tools for Structural Engineering," *Structural Engineering Review*. Vol. 5 (1), 1993.
57. Kuhn, T.S., *The Structure of Scientific Revolutions*. Chicago, IL: University of Chicago Press, 1962.
58. Kuhn, T.S., Postscript - 1969. In *The Structure of Scientific Revolutions*. Chicago, IL: University of Chicago Press. Enlarged 2nd Edition, pp. 174-210, 1970.
59. Kuhn, T.S., "Reflections on My Critics." In [63], pp. 231-278, 1970.
60. Kuhn, T.S., "Second Thoughts on Paradigms," Reprinted in T.S. Kuhn, *The Essential Tension*. Chicago, IL: University of Chicago Press, 1977.
61. Lakatos, I., "Falsification and the Methodology of Scientific Research Programmes," In [63], pp. 91-196, 1970.
62. Lakatos, I., *Proofs and Refutations*. Cambridge, U.K: Cambridge University Press, 1976.
63. Lakatos, I. and Musgrave, A. (eds.), *Criticism and the Growth of Knowledge*. Cambridge, U.K: Cambridge University Press, 1970.
64. Langley, P., Simon, H.A., Bradshaw, G.L. and Zytkow, J.M., *Scientific Discovery: Computational Explorations of the Creative Process*. Cambridge, MA: MIT Press, 1987.
65. Lansdown, J., *Design Studies*, Vol. 8 (2), pp.76-81, 1987.
66. Laudan, L., *Progress and Its Problems*. Los Angeles: University of California Press, 1977.
67. Lawson, B., *How Designers Think: The Design Process Demystified*. London: Architectural Press, 1980.
68. Lewin, K., Dembo, T., Festinger, L., and Sears, P.S., "Levels of Aspiration." In *Personality and the Behavior Disorder*, Hunt J.M. (ed.). New York: The Ronald Press, 1944.
69. Lindblom, C.E., "The Science of Muddling Through," *Public Administration Review*, Vol. 9, pp. 79-88, 1959.
70. Luckman, J., *Operational Research Quarterly*, Vol. 18 (4), pp. 345-358, 1967.
71. Maher, M.L., "A Knowledge-Based Approach to Preliminary Design Synthesis," *Report EDRC-12-14-87*, Carnegie Mellon University Engineering Design Research Center, 1987.
72. Maimon, O. and Braha, D., "A Mathematical Theory of Design: Representation of Design Knowledge (Part I)," *International Journal of General Systems*, Vol. 25 (3), 1997.
73. Maimon O. and D. Braha, "On the Complexity of the Design Synthesis Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 26 (1), 1996.

74. Manton, S.M., "Engineering for Quality," In *Taguchi Methods*, Bendell, Disney and Pridmore (eds.), IFS publications, 1989.
75. Masterman, M., "The Nature of a Paradigm," In [63], pp. 59-90, 1970.
76. Mostow, J., "Toward Better Models of the Design Process," *The AI Magazine*, Spring, pp. 44-57, 1985.
77. Mueller, R.A. and Varghese, J., "Knowledge-Based Code Selection in Retargetable Microcode Synthesis," *IEEE Design and Test*, Vol. 2 (3), pp. 44-55, 1985.
78. Murthy, S.S. and Addanki, A. 1987, "PROMPT: An Innovative Design Tool," In *Proc. of the 6th Nat. Conf. on Artificial Intelligence*, Seattle, WA.
79. Newell, A. and Simon, H. A., *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
80. Newell, A. and Simon, H. A., "Computer Science as Empirical Enquiry: Symbols and Search," (ACM Turing Award Lecture), *Comm. ACM*, Vol. 19 (3), March, pp. 113-126, 1976.
81. Nilsson, J.N., "Logic and Artificial Intelligence," *Artificial Intelligence*, Vol. 47, pp. 31-56, 1991.
82. Osborn, A.F., *Applied Imagination*. New York: Charles Scribner's Sons. 1963.
83. Paynter, H.M., *Analysis and Design of Engineering Systems*. Cambridge, MA: MIT Press, 1961.
84. Penberthy, J.S., *Incremental Analysis and the Graph of Models: A First Step Towards Analysis in the Plumber's World*, S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, 1987.
85. Pahl, G. and Beitz, W., *Engineering Design*. Berlin: Springer-Verlag, 1988.
86. Popper, K.R., *The Logic of Discovery*. London: Hutchinson (originally published, 1935), 1959.
87. Pugh, S., *Total Design*. New York: Addison-Wesley, 1990.
88. Pugh, S. and Smith, D.G., "CAD in the Context of Engineering Design: The Designers Viewpoint," In *Proceedings CAD*, London, pp. 193-198, 1976.
89. Pugh, S. and Morley, I.E., *Toward a Theory of Total Design*, University of Strathclyde, Design Division, 1988.
90. Rehg, J., Elfes, S., Talukdar, S., Woodbury, R., Eisenberger, M. and Edahl, R., "CASE: Computer-Aided Simultaneous Engineering," *AI in Engineering Design*, Gero, J.S. (ed.), Springer-Verlag, Berlin, pp. 339-360, 1990.
91. Ressler, A.L., "A Circuit Grammar for Operational Amplifier Design," *Technical Report 807MIT*, Artificial Intelligence Laboratory, 1984.
92. Rieger, C. and Grinberg, M., "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms," In *Proc. of the 5th Int. Joint Conf. on Artificial Intelligence*, pp. 250, 1977.
93. Rinderle, J.R., *Measures of Functional Coupling in Design*, PhD dissertation, MIT, 1982.
94. Rinderle, J.R., "Function and Form Relationships: A basis for Preliminary Design," *Report EDRC-24-05-87*, Carnegie Mellon University Engineering Design Research Center, Pittsburgh, PA, 1987.
95. Rittel, H.W. and Webber, M.M., "Planning Problems are Wicked Problems," *Policy Sciences*, Vol. 4, pp. 155-169, Reprinted in [21], pp. 135-144, 1973.
96. Roylance, G., "A simple Model of Circuit Design," *Technical Report 703*, MIT Artificial Intelligence Laboratory, 1983.
97. Rychener, M. (ed.), *Expert Systems for engineering design*. New York: Academic Press, 1988.
98. Schön, D.A., *The Reflective Practitioner*. New York: Basic Books, 1983.
99. Serbanati, L.D., *IEEE 9th International Conference of Software Engineering*, pp. 190-197, 1987.
100. Shina G.S., *Concurrent Engineering and Design for Manufacture of Electronics Products*. Van Nostrand Reinhold, 1991.
101. Siddall, J.N., *Optimal Engineering Design: Principles and Applications*. New York: Dekker, M., 1982.
102. Simon, H.A., "The Structure of Ill Structured Problems," *Artificial Intelligence*, Vol. 4, pp. 181-200, Reprinted in [21], pp. 145-165, 1973.
103. Simon, H.A., "Style in Design," In *Spatial Synthesis in Computer Aided Building Design*. Eastman, C.M. (ed.), John Wiley Sons, New York, 1975.
104. Simon, H.A., *Administrative Behavior*, 3rd Edition. New York: The Free Press, 1976.
105. Simon, H.A., *The Science of the Artificial*. Cambridge, MA: MIT Press, 1981.
106. Simon, H.A., *Models of Bounded Rationality*, Vol. 2. Cambridge, MA: MIT Press, 1982.
107. Simoudis, E. and Miller, J.S., "The Application of CBR to Help Desk Applications," *Proceeding of the 1991 Case-Based Reasoning Workshop*, Washington, DARPA, 1991.

108. Spillers, W.R. (ed.), *Basic Questions of Design Theory*. Amsterdam: North-Holland, 1972.
109. Sriram, D. and Cheong, K., "Engineering Design Cycle: A Case Study and Implications for CAE," In *Knowledge Aided Design*, Academic Press, New York, 1990.
110. Sriram, S., Stephanopoulos, G., Logcher, R. et al., "Knowledge-Based System Applications in Engineering Design: Research at MIT," *AI Magazine*, Vol. 10 (3), pp. 79-96, 1989.
111. Steinberg, L.I., "Design as Refinement Plus Constraint Propagation: The VEXED Experience," *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 830-835, 1987.
112. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.
113. Suh, N.P., Bell, A.C. and Gossard, D.C., "On an Axiomatic Approach to Manufacturing and Manufacturing Systems," *Journal of Engineering for Industry*, Vol. 100 (5), pp. 127-130, 1978.
114. Swartout, W. and Balzer, R., "On the Inevitable Intertwining of Specification and Implementation," *COMM. ACM*, Vol. 25 (7), July, pp. 438-440, 1982.
115. Sycara, K. and Navinchandra, D., "Integrating Case-Based Reasoning and Qualitative Reasoning in Design," In Gero, J. (ed.). Ashurst, Computational Mechanics Publishing, 1989.
116. Tong, C., *Knowledge-Based Circuit Design*, PhD dissertation, Stanford University, 1986.
117. Tong, C. and Sriram, D. (eds.), *Artificial Intelligence in Engineering Design*. Boston, Mass: Academic Press, 1992.
118. Toulmin, S., *Human Understanding*. Princeton, NJ: Princeton University Press, 1972.
119. Ullman, D., Stauffer, L. and Dieterich, T., "Preliminary Results of Experimental Study of the Mechanical Design Process," *Technical Report 86-30-9*, Oregon State University, C.S. Dept., 1986.
120. Ulrich, K.T., "Computation and Pre-Parametric Design," *Technical Report 1043*, MIT Artificial Intelligence Laboratory, 1988.
121. Vollbracht, G. T., "The Time for CAEDM is Now," *Computer-Aided Engineering*, CAD/CAM section, Vol. 7 (Oct.), pp. 28, 1988.
122. Winston, P.H., et. al., "Learning Physical Descriptions From Functional Definitions, Examples and Precedents," *Memo 679*, MIT, Artificial Intelligence Laboratory, 1983.
123. Wood, K.L. and Antonsson, E.K., "Engineering Design-Computational Tools in the SYNTHESIS Domain," *The Study of the Design Process*, A Workshop, Ohio State University, February, pp. 8-10, 1987.
124. Yoshikawa, H., "General Design Theory and a CAD system," *Man-Machine Communications in CAD/CAM*, Proceedings, *IFIP W.G 5.2, Tokyo*, North-Holland, Amsterdam, pp. 35-38, 1982.
125. Konda, S., Monarch, I., Sargent, P., and Subrahmanian, E., "Shared Memory in Design: A Unifying Theme for Research and Practice," *Research in Engineering Design*, Vol. 4 (1), pp. 23-42, 1992.
126. Subrahmanian, E., Konda, S., Levy, S., Reich, Y., Westerberg, A., and Monarch, I., "Equations Aren't Enough: Informal Modeling in Design," *AI EDAM*, Vol. 7 (4), pp. 257-274, 1993.
127. Pinch, T. K., and Bijker, W. E., "Social Construction of Facts and Artifacts," In Bijker, W. E., Hughes, T. P., and Pinch, T. W., *Social Construction of Technological Systems*, MIT Press, Cambridge, 1989.
128. Sriram, D., Wong, A., and Logcher, R., "Shared Workspaces in Computer Aided Collaborative Product Development," Technical report, *Intelligent Engineering Systems Laboratory*, 1991.
129. Sargent, P. M., Subrahmanian, E., Downs, M., Greene, R., and Rishel, D., "Materials' Information and Conceptual Data Modeling," *Computerization and Networking of Materials Databases: Third Volume, ASTM STP 1140*, Thomas I. Barry and Keith W. Reynard, editors, American Society for Testing and Materials, Philadelphia, 1992.
130. Danko, G. and Printz, F., "A Historical Analysis of the Traditional Product Development Process as a Basis for an Alternative Process Model," *EDRC report*, Carnegie Mellon University, 1989.
131. Clark, K., and Fujimoto, T., *Product Development Performance*, Harvard Business Press, 1991.
132. Woodforde, J., *The Story of the Bicycle*, Routledge and Kegan Paul, London, 1970.
133. Westerberg, A., "Distributed and Collaborative Computer-Aided Environments in Process Engineering Design," *EDRC report*, Carnegie Mellon University, 1996.
134. Bjork, B. C., "Basic Structure of a Proposed Building Product Model," *Computer Aided Design*, Vol. 12 (2), 1988.
135. Eastman, C., Bond, A., and Chase, S., "A Formal Approach for Product Model Information," *Research in Engineering Design*, Vol. 2, pp. 65-80, 1991.
136. Zamanian, K., Fenves, S. J., and Gursoz, E., "Representing Spatial Abstractions of Constructed Facilities," *EDRC report*, Carnegie Mellon University, 1991.

137. Hauser, J. H., and Clausing, D., "The House of Quality," *Harvard Business Review*, pp. 63-73, May-June 1988.
138. Bloor, D., "Wittgenstein and Manheim on the Sociology of Mathematics," *Studies in the History and Philosophy of Science*, Vol. 4, pp. 173-191.
139. Mulkay, M. J., "Knowledge and Utility: Implications for the Sociology of Knowledge," *Social Studies of Science*, Vol. 9, pp. 63-80, 1979.
140. Collins, H. M., "An Empirical Relativist Programme in the Sociology of Scientific Knowledge," in *Science Observed: Perspectives on the Social Study of Science*, K. D. Knorr-Cetina and M. J. Mulkay, eds., Sage, Beverly-Hills, pp. 85-113, 1983.
141. Collins, H. M., "The Seven Sexes: A Study in the Sociology of a Phenomenon, or the Replication of Experiments in Physics," *Sociology*, Vol. 9, pp. 205-224, 1975.
142. Collins, H. M., and Pinch, T. J., *Frames of Meaning: The Social Construction of Extraordinary Science*, Routledge and Kegan Paul, London, 1982.
143. DeMillo, R. A., Lipton, R. J., and Perlis, A. J., "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM*, Vol. 22, pp. 271-280, 1979.
144. Nam P. Suh, "Design and Operation of Large Systems", *Journal of Manufacturing Systems*, Vol. 14, no. 3, pp 203-213, 1995.
145. Ertas, A. and Jones, J. C., *The Engineering Design Process*, John Wiley & Sons, New York, 1995.
146. Dasgupta, S., *Design Theory and Computer Science*, Cambridge University Press, 1994.
147. Ferguson, E. S., *Engineering and the Mind's Eye*, MIT Press, Cambridge, MA, 1992.
148. Sipper, M. et. al., "A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems," *IEEE Transactions on Evolutionary Programming*, Vol. 1, No. 1, pp. 83- 97, 1997.
149. Ramsay, A., *Formal Methods in Artificial Intelligence*, Cambridge University Press, 1988.
150. Maimon, O. and Horowitz, R., "Creative Design Methodology and the SIT Method," Proceedings of DETCC97/DTM-3865, 9th International ASME Design Engineering Theory and Methodology Conference, 1997. Awarded Best Conference Paper.
151. Maimon, O. and Horowitz, R., "Sufficient Conditions for Design Inventions", to appear in *IEEE Systems Man and Cybernetics*, 1998.

CHAPTER 3

INTRODUCTORY CASE STUDIES

It is not enough to just read about the design process as discussed in this book to truly understand the role of each phase, and the evolution that causes transformation between a functional and a structural description of a device. One must experience it. This chapter attempts to provide experience through some examples of design processes. These five cases were chosen to give the reader insight into the design process (emphasizing the synthesis part), and the thinking of designers in different design situations. Evidence must be provided to determine whether or not the design process is evolutionary in nature. We believe that the evidence in the following examples is compelling, and serves as strong corroboration that the design process is evolutionary in nature.

3.1 ELECTRICAL DESIGN

3.1.1 DESIGN OF A "DATA TAG" [4]

The following example provides a brief description (for an in-depth description consult [5]) of the product "Data Tag" that was developed in the Flexible Manufacturing Cell at Tel-Aviv University (TAU CIMD Laboratory).

The Design Task

According to Shmilovici and Maimon [5]: "Data Tags (DT) are small electronic devices which can store large amounts of data in a compact and economic volume. Unlike ordinary devices, the contents of a DT can be read or modified by a computer without the need for physical contact between the DT and the computer."

In a production environment, such as the Flexible Manufacturing Cell at Tel-Aviv University, DTs are attached to the pallets that carry the parts to the robots; thus, providing a unique identity for each unit produced (rather than just controlling the machine centers). Installed on the system are two read-write heads; one in front of each junction of the long closed loop conveyor, which is composed of two smaller inner loops (there are two junctions that enable each inner loop to operate independently when needed). When a pallet arrives at a junction, the contents of the

DT is read and the local controller uses the information to make decisions; for example, whether the pallet is to stay in the inner loop or not.

The properties of DTs include their ability to store large amount of information that can be modified during the production cycle; their ability to resist many of the harsh conditions that are common on the factory floor; and their ability to be integrated in the production process. The effectiveness of DTs can be attributed to the ability to: incorporate product changes based on customer requests; enable to accompany the product during its processing stages; document the exact information about the processing stages that were given for each product, and thus enhancing statistical process control; monitor the quality of suppliers, customer service, accounting and inventory control; and enhance the Kan-Ban method for controlling Work In Process (WIP) by replacing the card that accompanies the product between work-stations [5].

The Evolution of the "Data Tag"

The Initial Requirements

The desire to apply DT technology emerges from the need for a "paperless production" process. The designers at the Robotics Lab at TAU came to utilize this technology by studying the production processes, the existing computing power in the processes where the DTs are to be used, and the possibility of adding requirements based on future production or customer needs. The initial specifications involved in the DT design are:

$\theta_{0,1}$: The DTs should conform to the protocols of the read-write heads that are attached to personal computers and read the contents of the DTs that arrive at a particular junction. The personal computers are also connected (via network) to the central computer; which transmits instructions and gathers information.

$\theta_{0,2}$: The data capacity of a DT should be about 200 bytes with a RAM battery that enables extended operation time, and small physical dimensions.

The original specifications are replaced (or modified) by the new specifications:

$\theta_{1,1}$: The DT should have the capacity to withstand harsh manufacturing conditions including dirt, high temperature, and corrosive materials.

$\theta_{1,2}$: DTs need to attach to the pallets that carry the parts to the robots.

$\theta_{1,3}$: The DT's battery should have low power working modes.

$\theta_{1,4}$: The DT needs to continuously operate for at least 900 hours with a disposable battery. Alternatively, it needs to continuously operate for at least 200 hours with a reusable battery.

$\theta_{1,5}$: The DT needs to handle transmission rates of 9600 baud.

$\theta_{1,6}$: The data on the DTs should include the communication protocol software between the DT and the local personal computers.

$\theta_{1,7}$: The DTs need to be manufactured in large batches. Thus, a printed circuit board technology is preferred.

$\theta_{1,8}$: The manufacturing cost of the DT should be less than 100\$.

The DT has to communicate with a read-write heads that are installed in front of each junction of the conveying system. To carry out this communication, a communication protocol between the DTs and the read-write heads should be defined. This communication protocol software should follow the following general properties¹:

$\theta_{1,9}$: Its data capacity should be bound to 8K.

$\theta_{1,10}$: It needs to withstand transmission rates of 9600 baud.

$\theta_{1,11}$: It should be easily modifiable in terms of software, hardware and the information capacity.

The Evolution of the Transmission Method

There are various types of read/write heads that can communicate with different forms of DTs, and are suitable for a variety of production processes and control demands. To reduce significantly the number of compatible transmission methods between the DTs and the local computers the following requirements are considered:

$\theta_{1,12}$: The technology should be cost effective.

$\theta_{1,13}$: It has to withstand high transmission rates.

$\theta_{1,14}$: It has to have low power working modes.

$\theta_{1,15}$: The parts count needs to be relatively low.

$\theta_{1,16}$: The operation needs to be simple.

$\theta_{1,17}$: The data on the DT includes a status variable that can indicate faults in the integrity of the data. If faults are detected the transmission can be reset.

$\theta_{1,18}$: The communication channel can support interference from manufacturing equipment and disturbances generated by other Data Tags in the vicinity.

There are several transmission methods that use different wavelengths from the electromagnetic spectrum: infrared light, radio frequencies, magnetic induction and electronic contact. Accordingly, the transmission range varies from zero for electronic contact; several millimeters to a few meters for most methods; and up to hundreds of meters for some forms of Radio transmission.

The designer may conclude that electronic contact such as Barcode technology can not cope with many of the DTs specifications. The process of reading the Barcode label is not reliable for many production processes, there is a need for close proximity between the read-write head and the label, and the reading station is relatively expensive. Several other transmission methods including the by-wire method were excluded for similar reasons.

The TAU designer finally arrived at two transmission methods. One method uses short range radio transmission, and does not need a battery as a power source. The read-write unit is composed of an antenna, which is connected to an electronic board that may be hosted in a personal computer. The other method uses infrared light for transmission and is battery powered. The main advantages are in its ability to withstand harsh manufacturing conditions that include dirt, high temperatures, and

¹ Here we focus on the physical design of the DTs. The evolution of the software and data structures design of the DTs and the local computers are detailed in [4].

corrosive materials; and in the simplicity of the read-write head that can be attached to any serial data channel. Since the adequacy of a design is determined solely with respect to the requirements prevailing at that stage, the foregoing transmission methods were critically tested against the requirements $\theta_{1,12}$ - $\theta_{1,18}$ and the infrared light transmission method was found to meet the requirements, i.e., there is a fit between design and requirements.

Concept Design

In the case of the DT design problem, the large number of options may initially be reduced by a decision to adopt a particular architectural style. The selected architectural style may not be the 'optimal' decision, but a 'satisfying' one. In our case, a high level conceptual solution was developed based on bus-architecture, as shown in Figure 3.1. To satisfy the low power working modes requirements ($\theta_{1,3}$), the designer adopted two working modes: Power-down working mode, which occurs when the pallets carry the parts between the read-write heads; and power-up working mode, which occurs when a pallet arrives at a junction. The contents of the DT is then read by the local controller. The various components and steps involved in the DT system are (see Figure 3.1):

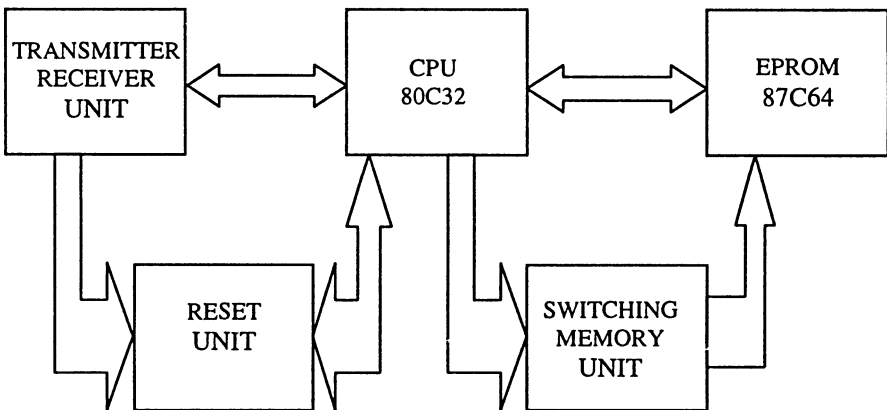


Figure 3.1 The Architecture of the "Data Tag"

- The CPU communicates with the local computer when a new part enters the communication range. After each processing stage, the CPU modifies the record on the DT (e.g., information regarding the last process taken, its duration and measurement results). The CPU also controls the system's units (e.g., the power source to the DT).
- The Memory unit is EPROM-type, and includes the communication protocol software. Each DT has a unique identification number, a local identification

number assigned by the local controller, a description of the current parts' process plans, and historical information that describes the parts' past processes, process times, and machines and any parameters measured after the process.

- The Switching Memory unit supports the CPU by controlling the connection between the power source and the memory unit. By disconnecting the power source from the memory unit, when the system switches to a power-down working mode, the current drain in the system can be minimized.
- The Transmitter-Receiver unit supports the CPU by carrying out communications with the local computer, and in switching the system to a power-up working mode when a pallet arrives at a junction.
- The Reset unit enables to produce the desired reset pulse based on the received signal.

Evolution of the CPU unit

The initial specifications (constraints) posed on the CPU are: it has to be integrated in an analog circuit ($\theta_{2,1}$); it should be able to be attached to any serial data channel without using a universal asynchronous receiver-transmitter unit ($\theta_{2,2}$); its part count should be relatively low ($\theta_{2,3}$); it has 4K bytes ROM and 256 bytes RAM ($\theta_{2,4}$); it includes timers ($\theta_{2,5}$). After considering the requirements $\theta_{2,1}$ - $\theta_{2,2}$, the designer decided to use INTEL 8051 family of micro-controllers. As soon as the selection was made, an additional constraint was posed on the design: the CPU should have power reduced mode ($\theta_{2,6}$). Therefore, the designer came up with the following processors based on CMOS technology:

1. 8051 - 4K ROM
128 byte RAM
cost = \$30
2. 8052 - 8K ROM
256 byte RAM
cost = \$40
3. 8031 - without ROM
128 byte RAM
cost = \$6
4. 8032 - without ROM
256 byte RAM
cost = \$8

As soon as the selection was made, an additional constraint was posed on the design (the CPU): it should be cost effective ($\theta_{2,7}$). Therefore, the designer decided to use the 80C32 processor. Each of the units in the DT's design plan had its own (evolutionary) design plan. For example, the memory unit design plan involved selecting the 87C64 memory unit that has 4K byte, has power reduced mode, is easily connected to the CPU, and is of EPROM type.

3.1.2 CONTROL LOGIC OF FLEXIBLE MANUFACTURING SYSTEMS

A flexible manufacturing cell (FMC) is a set of processing machines, a transportation system (e.g., a conveyor belt), a set of work-pieces, a system for storing parts in processes, and a control system. In a FMC the control logic would commonly be implemented on a small specific purpose computer called a Programmable Logic Controller (PLC). The typical PLC is programmed by using ladder logic diagrams or Boolean equations. In either case the execution of the program is the same, with all input values being scanned and all outputs being reevaluated once every cycle.

To illustrate, let us consider a simple design problem. For instance, a control mechanism that must be designed to achieve the following behavior in a flexible manufacturing cell. The flexible manufacturing cell consists of a transport line, which transports part from one NC machine to another. If the pallet carrier arrives at the unload station (denoted by a binary input I_3) and exactly one NC machine is holding a finished part (denoted by binary inputs I_1 and I_2 , respectively), then the transport line unloads a part from the machine (denoted by a binary output O). If both NC machines are done, then the transport line remains idle until the Fanuc robot unloads one of the finished parts to a temporary buffer.

If we approach this problem from a computer engineering perspective, then we would like to synthesize a switching circuit with the following behavior. If either I_1 or I_2 (but not both) and I_3 are activated then activate output O . The truth table for this behavior can be summarized in Table 3.1.

Table 3.1 The Behavior of the Logic Controller

OUTPUT	INPUTS		
	I_1	I_2	I_3
0	0	0	0
0	0	0	1
0	0	1	0
1	0	1	1
0	1	0	0
1	1	0	1
0	1	1	0
0	1	1	1

If this behavior is implemented with digital electronics, such as a Programmable Logic Controller, then the solution is to combine two Boolean gates (an EXCLUSIVE OR gate and an AND gate). The behavior of these two gates is shown in Table 3.2.

Table 3.2 Behavior of AND and EXCLUSIVE OR Gates

AND		
X	Y	O
0	0	0
0	1	0
1	0	0
1	1	1

EXCLUSIVE OR		
X	Y	O
0	0	0
0	1	1
1	0	1
1	1	0

The control logic design process is concerned with how to choose these two gates and how the structures are determined. Even if we consider the case where each gate has two inputs and one output, the number of possible structural solutions can be enormous. In addition, the behavior of a control logic solution changes with each structure. For example, from the simple behavior of the AND and EXCLUSIVE OR gates, two different new design solutions can be built. The first design alternative is shown in Figure 3.2. The second design alternative with the EXCLUSIVE OR and AND gates has parallel inputs to derive a two bit adder with carry as shown in Figure 3.3.

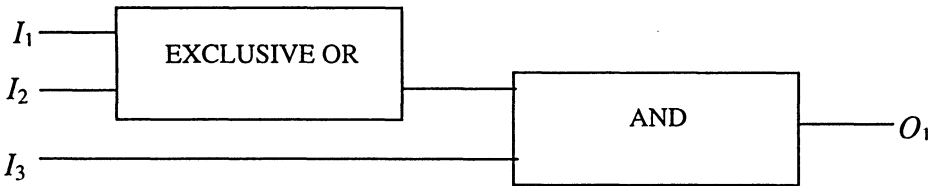


Figure 3.2 Design Alternative 1

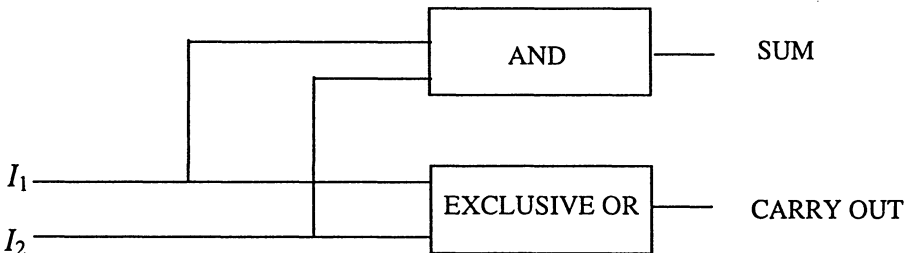


Figure 3.3 Design Alternative 2

3.1.3 SERIAL BINARY ADDER UNIT DESIGN

A serial binary adder unit design constitutes an important component in computer engineering design. The problem facing the designer is to synthesize a switching circuit using logical gates that implement the following behavior:

$$\text{Sum} = X + Y + \text{Carry-in} \pmod{2}. \quad (1)$$

$$\text{Carry-out} = (X + Y + \text{Carry-in} - \text{Sum})/2. \quad (2)$$

X and Y are the input bits. Carry-in and Carry-out are carry bits that represent the input and output carry respectively. The initial problem as stated is: given two binary inputs and a carry bit (received from previous calculations), devise a switching circuit using logical gates of OR, AND or EXCLUSIVE OR that computes both the new sum and carry bit.

A general approach to switching circuit synthesis is to use a decomposition process. This approach views a system as an interconnection of subsystems, and the components are modeled one for one. Together the interconnected system provides a design that satisfies the overall specification set. Hence, the designer devises a switching circuit, using logical gates of OR AND or EXCLUSIVE OR, that implements the behavior as described in Equations 1 and 2. Thus, the specifications are reformulated so that the design solution satisfies two functional properties, CARRY and SUM, as follows:

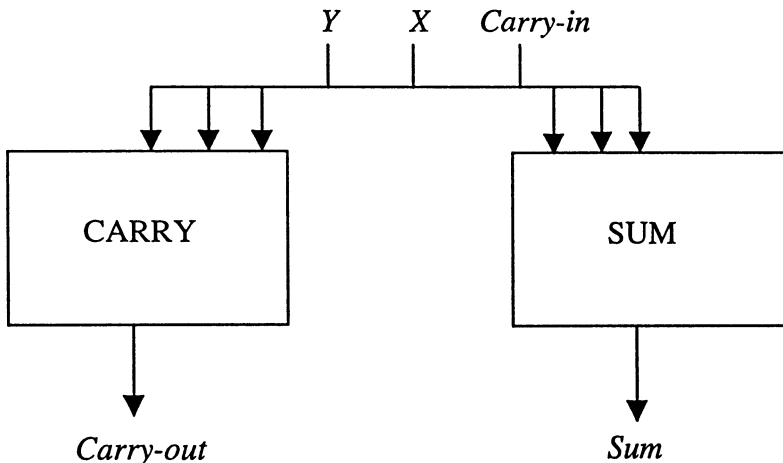


Figure 3.4 High-level Specification for A Serial Binary Adder Design

Applying any conventional procedure for simplifying Boolean expressions, the designer arrives at the following solution for CARRY and SUM (Figure 3.5a-b).

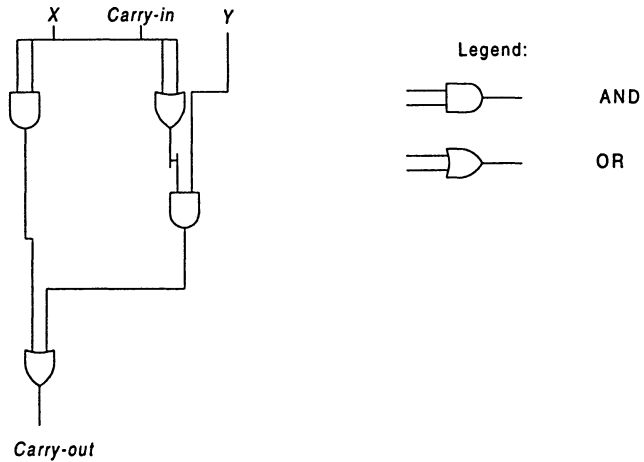


Figure 3.5(a) CARRY Unit Design

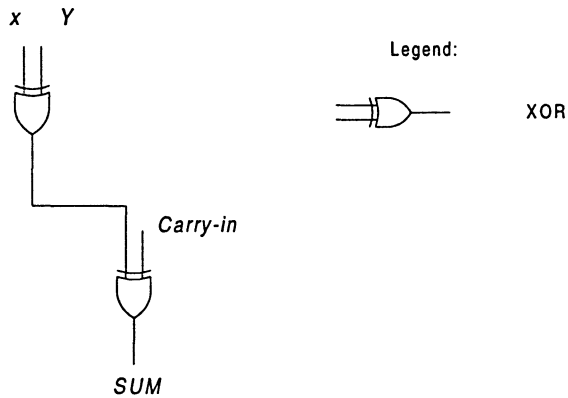


Figure 3.5(b) SUM Unit Design

3.2 MECHANICAL DESIGN

3.2.1 MECHANICAL FASTENERS DESIGN [1]

Of the methods commonly available for joining materials and parts *mechanical* fastening is the most versatile. The threaded method is classified as semi-permanent and the riveted or stamped method as permanent. *Adhesive* fastening requires a more

carefully controlled environment for its application and may be classified as a permanent joining method. *Soldering* and *welding*, which are permanent methods of fastening, are more versatile than adhesives but less versatile than mechanical methods.

The design of mechanical fasteners is a common design problem in mechanical engineering. The incredible variety of mechanical fasteners (over two million different kinds) has caused some manufacturers, who have large assemblies, to carefully re-evaluate the function of each fastener for better standardization. Fasteners may be divided into five main types: threaded, rivets, washers and retaining rings, pin, and quick-operating.

The proposed problem is to design a new fastener according to certain given *specifications* (including service capability, performance, mechanical, and physical specifications) based on the knowledge that designers already have from the existing fasteners. The result of the fasteners design process is a set of potential fasteners; each of which consists of a set of *structural* attributes.

The *structural description* of every fastener is specified in terms of properties that can be observed. The fastener is composed of six components that are common to all fasteners: drive, head, shoulder, body (shank), tail, and tip (point). The drive is the structure that forms the interface to the fastener actuator. The body is the structure which occupies the hole in the plate. The head and tail are the structures that provide the tensile forces to the fastened plates when the plates are pulled apart. The point is the structure that enters the hole first when the fastener is actuated.

Besides these five essential structural features, fasteners can be specified by the nominal size (fraction diameter), number of threads per inch, thread form (the configuration of the thread in an axial plane), and thread fit (that specifies the allowance between the nut and the bolt). Fasteners may also be identified by the materials from which they are formed. Fasteners are made of dozens of materials such as ferrous, non-ferrous, and non-metallic (plastics). Such variety is necessary with the almost limitless combinations of conditions in which fasteners are expected to perform.

The *functional abstraction* of a design is usually determined by the structural hierarchy. For instance, the function of a fastener is to hold two plates together. This is done by the fastener that transmits the load to the fastened parts. The fastener is actuated with the drive, and is inserted in the hole. In addition, the function of the fastener may depend on a second device such as a nut. Other functional attributes that are commonly used to describe the function of a fastener are the physical, mechanical, and performance characteristics. *Physical* properties are inherent in the raw material and remain unchanged. Such physical properties as density, coefficient of thermal expansion, electrical resistance, thermal conductivity, and magnetic susceptibility all have importance and frequently dictate fastener material selection. *Mechanical* properties identify the reaction of a fastener to applied loads. Rarely are the mechanical properties of the fastener equal to those of the raw material from which it is made. Examples of mechanical properties are tensile strength, yield strength, hardness, and ductility. *Performance* properties are functional design features built into the fastener in order to satisfy service application criteria. Criteria

for service application include retractability, required modifications to the hole, degree to which access to the back of the second plate is required, ease with which the fastener can be actuated, insertion-case, adjustability, locking ability, prevailing torque, sealing, and the precision with which the fastener laterally locates the first plate with respect to the second plate.

The functional attributes are derived directly from the structure of a fastener, or may include other sub-functions. The relation is usually described as causal and governed by the physical laws, as illustrated in the following:

- The key to proper fastener selection (that is selecting its five essential structural components) is knowing what job (mechanical as well as service application) the fastener is expected to do and then making the specifications accordingly. For example, permanent fastening is often done with *rivets*. Sizes are designated by body diameter and length and range from those used in bridges to those used in small toys and watches. *Blind rivets* are used to fasten together plates when only one side of the assembly is accessible. *Quick-operating* (or quick release) rivets are used to fasten sheet-metal parts that must be spring pressured, periodically opened, yet securely closed. *Pin* rivets are used as a locking collar to shafts and hinges.
- The strength (a functional attribute) of mating threads depends on adequate engagement or overlap of the thread in the transverse direction (a structural attribute). Ample evidence shows the desirability of having plenty of “breathing room” between mating threads to accommodate local yielding, thread bending, and elastic deformations [1]. In general, medium thread fit with low- and medium-strength materials are used for designing high strength fasteners. Alternatively, the designer may use a closer thread fit with materials of higher tensile strength and lower ductility.
- Fasteners are available in almost any material. Common considerations for material selection are environment, weight, load characteristics, cost, reuse, and life expectancy. Aluminum is used where a high tensile strength-to-weight ratio is required. It is also used for corrosion resistance and appearance. Brass in a cold-drawn state has a greater tensile strength than mild steel and has a higher resistance to corrosion. It also has a high lustrous finish and is non-magnetic. Stainless-steel fasteners are used where problems of corrosion, temperature, and strength exist. Plastic fasteners have excellent corrosion resistance. Salt water has no effect on nylons and mineral acids have no effect on polyvinyl chloride. Plastics are recommended where good thermal and electrical insulators are required. However, plastics are not recommended where high tensile, operating temperatures or shear strengths are required.
- Some functional attributes can be complex and not easily inferred from one structural attribute. In this case, other sub-functions may be included in order to simplify the process. For example, the strength of the fastener (its ability to support and transfer loads) is determined by the tensile strength, fatigue strength, hardness, ductility, toughness, shear strength, torsional strength, stress rupture, and impact strength. Tensile strength is determined by the material and tensile stress area. Fatigue is determined by the material, head size, thread size, threaded

length, fillet, fabrication and surface structure and so on. These properties, in turn, are determined by the structural attributes of the fasteners (including dimensions and materials).

3.2.2 SUPERCRITICAL FLUID CHROMATOGRAPHY (SFC) DESIGN [6]

This problem is related to a Supercritical Fluid Chromatography (SFC) design. SFC facilitates the analysis of complex, high-molecular-weight or thermally-labile mixtures (such as polymers, pharmaceuticals, foods, petrochemicals, pesticides). SFC utilizes a highly compressed gas, which is above its critical temperature. At the critical point, the vapor and liquid phases of substances have identical temperatures. The properties of supercritical fluids lie between those of gases and liquids. This facilitates the easy transport of analytes through the chromatographic column. The key to the effectiveness and versatility of SFC is the ability to change the density of the mobile phase, which changes the fluid's chromatographic properties (including solution, diffusion, and viscosity). A high level schematic view of SFC is shown in Figure 3.6.

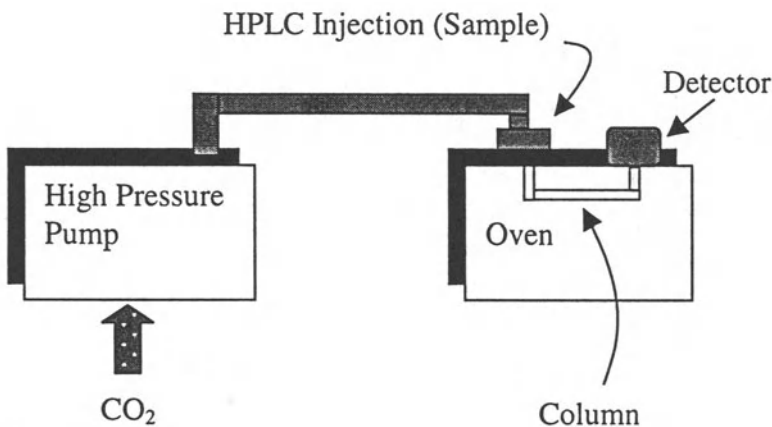


Figure 3.6 High Level Schematic View of SFC (adapted from [6])

Following is a brief sketch of the evolution of the SFC. By studying the literature and understanding the chromatographic process, designers arrive at a tentative design solution that consists of pump, control and oven units. The oven unit is further divided into injection, oven and detector units. This tentative design involves the following process: carbon dioxide is compressed to the desired pressure and pumped into the column and the detector; the sample mixture is then injected through the injector to the column with the carbon dioxide; and separation takes place. The detector controls the pump unit by detecting the amounts of the different compounds in the sample mixture. The next step is to design the various modules: pump, oven, injection device, control device, and detector.

Let us examine the (partial) evolution of the design pump unit. The initial design

specifications as stated are: (1) it has to be pulseless; (2) it should be able to compress carbon dioxide. By invoking a large knowledge-base the designer in our case selects a syringe pump. After choosing the type of pump, an additional specification is derived: (3) the number of experiments per filling should be optimized. This specification is modified to the following: (4) the pump must be large; and the pump motor should generate high speeds, have high torque, and be highly accurate. Based on previous knowledge, the designer refines the specifications regarding the pump to include the design of its subunits (e.g., the power source, cylinder, gear, motor, piston rod). For example, in order to generate high speed, torque, and accuracy; the designer selects a stepper motor.

3.2.3 GEAR BOX DESIGN (WORMGEAR REDUCER)

A machine is a collection of parts designed to transmit and modify motion and force. The moving parts of a machine are connected such that a change in the position of one piece is likely to change the position of the others. Each part of a machine must be in contact with at least one other part. Two parts that are in contact, and may have relative motion, form a pair. Two elements of consecutive pairs may be connected together by a link. An assemblage of pairs connected by links constitute a *kinematics chain*, a *mechanism*, or a *gear*. Because they are capable of transmitting motion and power, gears are among the most important of all mechanical components. The remainder of this section illustrates some of the issues involved in the development of a gear box.

Confrontation

Suppose that you are an engineer at the Sears & Roebuck Co. The director of the Product Development Department asks you to design a mass production device that can be used for lifting light objects or opening garage doors in family homes or small warehouses.

The specifications for the device are not clear enough for you to start the design. You must speak with the Director for more details. Based on your experience, the device he has asked for is called a "Hoist." Unlike hoists used in industries and construction sites (KB), this one should be small, light, and affordable (low cost). Considering most moving and loading tasks in family homes and small warehouses (KB), the capacity range for the hoist should be around 100 to 150 lbs.

Based on hoists used in industries and construction sites, the hoist should also include a cable, drum, coupling, brake, reducer and crane motor. The working principle of the hoist is shown in Figure 3.7.

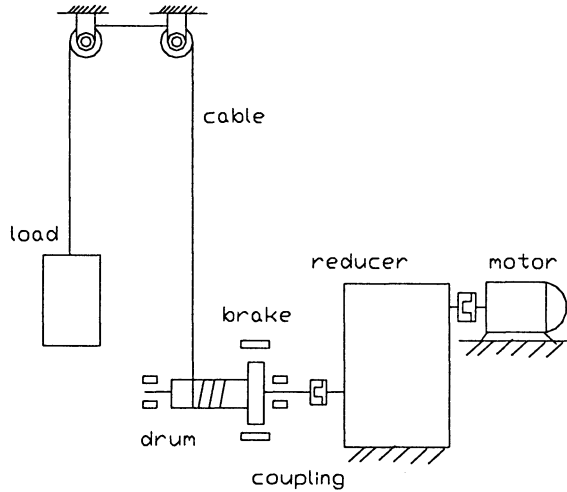


Figure 3.7 The Working Principle of the Hoist

Problem Formulation

Suppose that we are going to select the drum, cable, and electrical motor directly from the market. For general use (in small warehouses), the lifting speed should not be very fast ($V = 30$ feet per hour should be suitable). To raise a load of $F = 150$ lb., a wire cable must have at least 10 times the load 150 lb.; e.g., 1,500 lb. From the catalog, we select the 1/4 inch diameter wire cable (breaking strength 4,000 lb). To insure a longer life for the rope, overbending the rope must be avoided. Thus, the drum diameter should be large. According to the engineering recommendation value, the drum should be 15 to 20 times the rope diameter. Thus, we choose $d_{dr} = 3.75$ inches drum diameter. Most industrial electric motors have a synchronous (no load) speed of $n_m = 1,500$ rpm. The basic requirement of designing a reducer is replaced with the following specifications and constraints:

$$\text{Input speed:} \quad n_i = 1500 \text{ rpm (motor speed)} \quad (1)$$

$$\text{Output reducer speed :} \quad n_o = \frac{12V}{\pi d_{dr}} = \frac{12 * 30}{3.14 * 3.75} = 30.57 \text{ rpm} \quad (2)$$

$$\text{Reduction ratio:} \quad R = \frac{n_i}{n_o} = \frac{1500}{30.57} = 49.067 \approx 50 \quad (3)$$

Output power:
$$P_o = \frac{FV}{33,000} = \frac{150 \cdot 30}{33,000} = 0.136(\text{hp}) \quad (4)$$

Duration: 2000 hours per year for 5 years with a reliability of 90%

Overall dimension: 8" x 8" x 8" (L x W x H)

No special material or special machining process should be used.

Design Concepts

Many different types of reducers can be designed. We can start by drawing some preliminary sketches (conceptualizations) as shown in Figure 3.8. Belts, chains, and worm gears, and worm gears, and worm gears can be considered the means of transmission.

Many criteria have to be considered (including the weighting factor associated with each criteria):

1. Production (including material and machining costs) and operation costs (0.1);
2. Overall dimensions and weight (1);
3. Convenience of use and maintenance (0.7);
4. Reliability and endurance (0.4);
5. Simplicity (0.8).

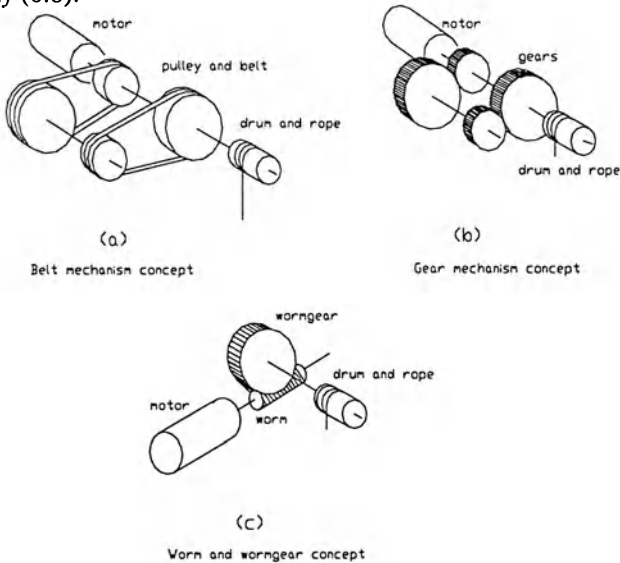


Figure 3.8 Sketch of the Design Concepts

Table 3.3 shows the rating of concepts with respect to criteria (lower is better):

Table 3.3 The Rating of the Design Concepts

	V belt & pulley	chain	spur gears	worm & wormgear
complexity of the structure	simple (1)	simple (1)	simple (1)	simple (1)
material	steel (1)	steel (1)	steel (1)	steel and bronze (1)
overall dimension	large (6)	medium (4)	compact (3)	small (1)
transmitting accuracy	low (6)	high (2)	high (2)	high (2)
mechanical efficiency	0.7~0.9 (2)	0.8~0.9 (1)	0.8~0.95 (1)	0.45~0.85 (3)
machining equipment	lathe (1)	lathe, mill (2)	lathe, mill or hob (4)	lathe & hob (2)
machining precision required	low (1)	medium (2)	high (3)	high (3)
assembly difficulty	easy (1)	easy (1)	easy (1)	easy (1)
noise level	low (1)	high (4)	medium (2)	medium (2)
maintenance	change belt (4)	lubrication (6)	check oil (1)	check oil (1)

The belt concept is the simplest transmission mechanism since pulley and shaft can be turned on a lathe. Since the power is transmitted by friction, the ratio is unstable when the load changes. Moreover, a certain distance must be maintained, between two wheel centers, to insure that the belt cover angle is greater than 120 degrees. This makes overall dimension of the design rather large.

The spur gear mechanism can obtain accurate transmission ratios, and results in a more compact design. However, a special gear generating machine (such as hob) is needed.

For both belt and spur gear mechanisms, at least two or three stages are needed to obtain a reduction ratio of 1:50. This means that more parts need to be included in the design. The design thus becomes more complicated in structure and larger in size.

The worm and wormgear mechanism is widely used for high ratio speed change. It can reach a ratio of ~ 30:100 in a single stage. This mechanism also has a simple structure and small dimension. Thus, comparing these alternatives, we decide to select the worm and wormgear concept.

We will not pursue the development of the wormgear reducer further except to note that the gear box was designed in detail and manufactured in the ADMS Laboratory at Boston University. During this process, we designed the structure of the transmission parts (worm and wormgear), shaft, bearings, casing (box), keys, couples, 'O' rings, screws, and the method of lubricating a sealing. The specifications

(constraints) considered included strength, rigidity, reliability, wear, friction, cost, ratio requirements, and space limitation. In chapter 17, we illustrate the development of the gear box including its complexities.

3.3 FLEXIBLE MANUFACTURING SYSTEM DESIGN [2]

3.3.1 WHAT IS A FLEXIBLE MANUFACTURING SYSTEM?

A Flexible Manufacturing System (FMS) can be defined as a computer-controlled configuration of semi-independent work stations with a material handling system designed to efficiently manufacture multiple parts at low to medium volumes. Figure 3.9 shows a conceptual drawing of a FMS. The definition and illustration highlight the three essential physical components (structural attributes) of the FMS:

- Standard numerically controlled machine tools;
- A conveyor that can move parts and tools within the network of machines and workstations;
- A control system that coordinates the functions of machine tools, part-moving elements, and work-pieces.

In most FMS installations, incoming raw work-pieces are loaded onto pallets at a load station that is set apart from the machine tools. They then move via the conveyor material handling system to queues at the CNC machines where they are processed. The flow of parts in the system is directed by the control computer, which acts as the traffic coordinator. In properly designed systems, the holding queues are seldom empty, i.e., there is a work-piece waiting to be processed when a machine becomes idle. When pallet exchange times are short, the machine idle times become quite small.

The number of machines in a system typically ranges from 2 to more than 20. The conveyor network system may consist of carousals, conveyors, carts, and robots. The important aspect of these systems is that the machine, conveyance, and control elements combine to enhance productivity and maximize machine utilization without sacrificing flexibility.

FMS technology has a relatively brief history. The original concept emerged in the mid- to late-1960's. It was a logical outgrowth of progress in applying numerical control, particularly attempts at factory-wide direct numerical control. During the 1970's, a number of FMSs were installed, and following a problem-shakedown period typical of new technology they became operational. Today, not only is the technology proven, it is increasingly available. A growing number of machine-tool builders are able to supply complete systems.

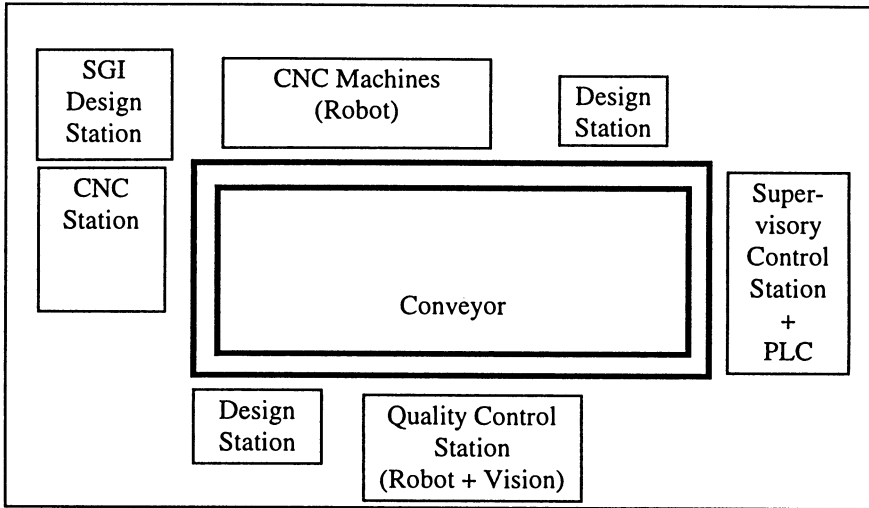


Figure 3.9 Schematic Configuration of a Flexible Manufacturing System

3.3.2 FMS CONFIGURATION DESIGN ISSUES

Once parts and machine types have been selected, it is possible to proceed to the system design process. There are several functional attributes involved:

- **Flexibility** - flexibility in an FMS has many aspects. The most important is the random-processing capability which allows multiple parts in the system at one time. Another type of FMS flexibility is *fault tolerance*. FMSs can continue to operate almost normally when there are machine failures. This is done by other machines “covering” for the one that is out of service. The ability to operate virtually untended is another type of FMS flexibility. Maintenance and part loading can be performed during the first shift, while much of the actual production occurs during the second and third shifts.

Desired flexibility affects FMS design. If an untended second- and third-shift operation is preferred then some automatic, in-line inspections may be needed. Very high system availability may not only require more reliable machines, material handling system (MHS), and computer but also back-up computers, duplicate machine types, and alternative routings in the MHS. Flexibility affects other design features as well. If an FMS is expected to handle a very wide range of part types, the machines will have fairly general characteristics such as five-axis capability, reasonable precision, more than minimum horsepower, etc. A desire to accommodate several parts simultaneously (without batching) may demand large storage capabilities. A related aspect is the expandability of the system to accommodate future increases in demand. The degree of expandability desired will affect the choice of MHS and the arrangement (layout) of equipment in the configuration.

- *Process Planning* - there are two areas of critical importance: the loading approach used for each part and the selection of cutting tools. In a FMS, it is important to minimize both manual and automatic handling of the part. Careful attention to loading designs can help. The use of window-frame and pedestal-type loading allows the greatest amount of access to a part when using either four- or five-axis machining centers. The system should be designed to have only one load/unload sequence. Minimizing the number of load/unload steps can be done by matching machine axes. For example, if vertical turret lathes (VTLs) are used for rotational work then vertical machining centers can be used for prismatic work.
Tool storage capacity needs can be minimized by standardizing the tool selection. For example, a standard 2- or 3-inch diameter shell mill could be used for all milling operations, except where restricted by corner radius or pocket size.
The FMS feeds and speeds that should be used depend on the rigidity of the part, the ability of dedicated pallets to hold the parts better than standard component-built pallets, and the use of new machine tools (perhaps with adaptive control) to maintain optimum cutting parameters.
- *Precision* - if high-precision machining is required, a high precision machine must be included in the system. Alternatively, the designer may be willing to customize general-purpose equipment to obtain the desired accuracy (if the cost can be justified). Finally, the designer may consider environmental control to obtain the desired accuracy. The environmental control specification derives the following structural features: temperature control of the FMS area, part/fixture/pallet temperature soaking, and coolant temperature conditioning. It may also derive temperature control of the inspection equipment.
- *System Availability* - machine availability (usually expressed as a percent) is the time during which a machine is not failed, i.e., the time it would be processing an available part. The availability is also called the "up-time" percentage. System availability is the percentage of time that none of the system components are failed, i.e., all machines, controllers, computers, the MHS, etc. are "up". The average availability of FMS components must be high if the average system availability is to be high. Therefore, a common design rule is to incorporate redundancy in the system. Thus, a system may contain two or more machines of each type, or it may have machines backing up dissimilar machines, e.g., a machining center might substitute for a multiple-spindle machine. Redundancy in the MHS implies multiple part carriers (e.g., carts) and multiple paths between stations (in case certain links fail). Obviously, there are tradeoffs between system redundancy, system complexity, component reliability, capital cost, and lost production cost.
- *Material Handling System* - The simplest MHS consists of a person and a cart, manually moving pallets and parts from machine to machine under computer direction. This manual system will work for small FMSs, where the distance between machines is short and parts are relatively small and light. To reduce machine waiting times, a shuttle loader can be added to each machine tool.

However, for larger systems and/or heavier parts, automatic MHSs are more applicable. These systems consist primarily of carts, conveyors, or robots that automatically transport pallets to and from shuttle loaders. If a loader is full, the pallet will circulate in the MHS, wait in front of the machine, or go to an off-line buffer storage area. Usually, a person is required to load and unload parts at the load/unload stations, but the rest of the system is computer controlled.

- *Tool Storage Capacity* - Machines with large tool-changer storage capacity are generally chosen because they reduce the need for production batching, and they can facilitate the processing of parts rerouted from failed machines.
- *Off-line versus On-line processes* - Several processes; such as very high accuracy machining, washing, inspecting, stress relieving, heat treating, deburring, finishing, marking and assembling, can be done off-line. If the cost of sending parts off-line and then returning them is high (especially if they must be unloaded from and reloaded to their pallet), then some of these processes could be on-line.

The design of an FMS configuration (guided by the above listed functional requirements) is a three-step process. The steps are:

1. *Estimate the work content of the selected parts:* (1.1) develop FMS loading process for the selected parts; (1.2) plan the process for each part in detail, constrained by the limited tool capacity of the FMS; (1.3) estimate production requirements for each part; (1.4) calculate part cycle times and tool usage.
2. *Design configurations:* (2.1) select specific vendors' equipment for each machine class; (2.2) estimate the minimum number of machines (spindles) needed for each machine class; (2.3) modify the estimated number of machines to account for shop and system efficiency, tool storage capacities, and machine redundancy desires; (2.4) add desired non-machining processes, such as automated inspection, material handling system, etc.; and (2.5) develop variations on the design configuration.
3. *Evaluate candidate FMS configurations:* (3.1) simulate the operation of each configuration; (3.2) improve configuration designs until each provides satisfactory performance measures or is rejected; (3.3) examine and evaluate intangibles such as flexibility, accuracy, etc.; and (3.4) choose the configuration that best satisfies both tangible and intangible requirements.

3.4 DISCUSSION

In this section we discuss some observations derived from the design case studies, and point out the direction of our future work. The design case studies reported in this chapter illustrate the complexity of the design process. While considering the design synthesis problem in the engineering realm, the cognitive aspects of designers cannot be ignored. Designers use the physical laws and also a priori designs based on their experiences. Hence, a good design synthesis should consider both these aspects.

The problem with studying design is understanding the relationship between function and structure. The function of a device is its purpose or intended use. A functional description consists of a set of functional attributes that might be characterized by a truth table, a performance curve, or a set of verbal phrases. The structure of a device is the information about the interconnection of modules organized either functionally (how the modules interact) or physically (how the modules are packed). A structural description consists of a set of structural attributes, which might be communicated through a drawing, model, or a direct implementation of the device itself. There are functions that are directly derived from the structure of a device (by different combinations of structural attributes). Other functional attributes may be more complex, and can not be inferred from one observable structural property.

Design is the transformation between a functional and structural description of a device. Design begins with some goal specifications of functional attributes. It becomes the designer's task to ascertain the causality for specific behaviors that lead to the pre-defined functionality, and then to generate a corresponding structure. Physical laws govern the functionality of the device. Hence, structural descriptions should be related to physical laws, and the physical laws should be either conjuncted or disjuncted to interpret or generate behavioral descriptions through causal reasoning.

The design case studies reveal a simple unifying notion: that the design process can be viewed as an alternation between specifications (functional attributes) and structural attributes. The alternation is performed in order to remove discrepancies between the behavior derived from the current structural attributes, and the current unsatisfied specifications of functional attributes. Only information, rules, and laws included in the designer's knowledge-base can be used in the process. The design process stops when adaptation is achieved. However, the process may resume if the specifications are subsequently altered, which disturbs the previous state of adaptation.

The search for design solutions is carried out by generating ideas, either intuitively or systematically (e.g., concepts of reducers, material handling systems, or "smart card" architectures) for the most appropriate methods at a given phase. The aim in seeking many solutions at a given phase is to explore the 'space' created by the great number of theoretically possible solutions. It is seldom possible to examine all solutions, as they are often innumerable. Such design problems are referred to as NP-complete (NPC), or Non-deterministic Polynomial time Complete problems [3]. The CPU time required to solve an NP-complete problem, based on known algorithms, grows exponentially with the "size" of the problem. There exists no polynomial time transformations for NPC problems nor are there any polynomial time algorithms capable of solving NP problems; therefore, these problems are considered to be "open" or unresolvable problems.

Despite the size of the 'solution space,' it be examined thoroughly so that it includes the main solution types. Only then can we say with reasonable likelihood that a satisfying solution can be chosen. In order to choose a solution we often represent designs more abstractly than is required for detailed design. More

specifically, we present a design formalism that explicitly represents the fundamental strategies and mechanisms in the design, i.e., the content or the semantics of the design. Through the realization of such a syntactic and semantic formalism, it becomes possible to articulate the essential concepts of design, and to model the artifacts and process of design across design disciplines.

The design case studies suggest that an inheritance scheme for data abstraction would be highly useful in making the design synthesis problem tractable. To achieve this goal, some of the design synthesis mechanisms described in subsequent chapters are implanted into programs, and constitute an intelligent computer-aided design system called CADAT (Case-based Design Advisory Tool), which is as introduced in chapter 10. When there is a need for a new design, the design specifications are usually given as functional or behavioral attributes. The CADAT system uses its knowledge of existing designs to link these attributes to the functional descriptions. This is done through the system hierarchy until they are linked to the physical components. The system interactively progresses through all the design specifications, and calls for user judgment at each design cycle. This enables participation in the construction of all the physical components that satisfy the initial design specifications.

REFERENCES

1. Blake, A., *What Every Engineer Should Know about Threaded Fasteners*. New York: Marcel Dekker, 1986.
2. *FMS Handbook*, CSDL-R-1599 U.S Army Tank Automotive Command under contract No. DAAE07-82-C-4040, 1983.
3. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1979.
4. Maimon et. al., "Electronic Data Storage on a Conveying Pallet," *Technical Report, The Robotics Systems Laboratory*, Tel-Aviv University (1990).
5. Shmilovici A., and Maimon O., "Suggested Uses for "Data Tag" Technology in Distributed Control of Flexible Manufacturing Systems," *Technical Report*, Department of Industrial Engineering, Tel-Aviv University (1993).
6. Sriram, D. and Cheong, K., "Engineering Design Cycle: A Case Study and Implications for CAE," In *Knowledge Aided Design*, Academic Press, New York, 1990.

PART TWO

FORMAL DESIGN THEORY (FDT)

CHAPTER 4

REPRESENTATION OF DESIGN ARTIFACTS

This chapter introduces the first part of Formal Design Theory (FDT): representation of design artifacts. The main goal is to lay out a domain independent modeling of design artifacts. The “minimalist” reader may skip over section 4.2.3, which provides an analysis of the artifact representation scheme, and proceed directly to the next chapter.

4.1 INTRODUCTION

The artifact space (alternately the attribute space, structure space, or the design space) is based on the following hypotheses:

Hypothesis 4.1 (Entity-Relational Knowledge Representation): an artifact representation is built upon the multiplicity of modules (attributes) and relationships among them.

Hypothesis 4.1 supports the stage of conceptual design. At the conceptual design stage, continuous properties are usually discretized. Although discretization might cause the design evaluation process to be imprecise, in most situations the aim of the conceptual design stage is to generate a variety of alternative concepts (described in terms of attributes or the structure of the artifact) that will meet a required set of specifications early in the project rather than to produce detailed engineering drawings. Therefore designers, at this stage, look for a conceptual design that will meet a pursued set of specifications, applying a set of modules

Hypothesis 4.2 (Nested Hierarchical Representation): the design of any complex artifact can be considered at various abstraction levels. The general direction of design is from more abstract to the less abstract levels. A design at any particular level of abstraction is a description of an organized collection of constraints (such as various structural, cause-effect, functional, and performance features) that are to appear in the physically implemented design.

These premises lead us directly to establish the artifact space as an *algebraic*

structure. An artifact is represented by a pair $\langle M, C \rangle$. M stands for the set of modules (or *attributes*) which the artifact is comprised of; and C denotes the set of relations that represent the relationships among the modules. In order to capture the essence of design as stated by Hypothesis 4.2, a hierarchical construction of artifacts from other artifacts is also developed. Consequently, the general set of modules is classified into *basic* and *complex modules*. Basic modules represent entities that can not be defined in terms of others. Complex modules are defined hierarchically in terms of other modules, where the effects of their interactions are captured.

To motivate the proposed generic artifact representation scheme, consider two typical examples:

1. In the conceptual design of a certain power transmission unit, the synthesis problem is broken down into sub-designs that include the design of speed reducers, clutches, motion converters, etc. Furthermore, the design of a speed reducer can be broken down into sub-designs corresponding to different types of speed reducers, such as gears, chains, etc. The design solution embedded in this tree structure takes the form of a path along the branches (which defines the interaction among modules), or may be simply an individual leaf of the tree.
2. Another example, where modules seem less structured, concern computer program design. Given an implementation of an algorithm in any language, it is possible to identify all the *operands* (modules), defined as variables or constants, that the implementation employs. Similarly, it is also possible to identify all of the *operators*, defined as symbols or combinations of symbols, that affect the value or ordering of an operand. The concept that an algorithm consists of operators and operands, and nothing else, is most easily verified by considering simple digital computers whose instruction format consists of only two parts: an operation code and an operand address. Following the stepwise refinement principles of program design, first proposed by Dijkstra [38], modules at each stage of the refinement procedure can be identified as statements expressed in pseudocode. The final stage will produce a program expressed in a compatible programming language.

Although these examples provide no indication whatsoever that designers have 'modules' in mind, it supposes to demonstrate the descriptive competence of the model introduced in this book. The trend of using 'object oriented' methods in large computer software projects also supports the foregoing postulates.

Any prescriptive approach to design should have some ability of forward search (from basic components toward complex components) and backward search (from complex components toward basic components). To perform these search strategies, the current chapter rigorously defines three types of artifact space operators, and explores their characteristics. These operators are able to manipulate transitive relationships among several relations, and perform any search tasks on a given artifact space. Two central types of operators are defined, termed as *Composition* (undertake composition tasks) and *Decomposition* (undertake decomposition tasks) operators. *Integration* operators combine both.

The chapter is organized as follows: Section 4.2 presents the main part of the chapter. Section 4.2.1 is devoted to a formal statement of the design space. Section 4.2.2 presents several examples. In Section 4.2.3 some properties of this representation are elaborated by means of several design-space operators. Finally, Section 4.3 concludes with some general remarks and directions for future work. For convenience, it was deemed to present all proofs in the appendix.

4.2 MODELING THE ARTIFACT SPACE

In this section we gradually construct a representation scheme of design artifacts.

4.2.1 THE BASIC MODELING OF DESIGN ARTIFACTS

Definitions 4.1: Basic Notations

1. The collection of all subsets of a given set Z is denoted by $P(Z)$. For example, if $Z = \{a, b, c\}$, then $P(Z) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, Z\}$.
2. The finite n -tuples (briefly, tuples) are denoted between angle brackets, for example, $\langle a, b, c \rangle$. The symbol $\langle \rangle$ means the empty tuple.
3. The tuple set of a set Z is denoted by $T(Z)$. The elements in $T(Z)$ are ordered tuples whose components belong to the set Z . The set $T(Z)$ contains all permutation of possible tuples. For example, if $Z = \{a, b, c\}$ then $T(Z) = \{\langle \rangle, \langle a \rangle, \langle b \rangle, \langle c \rangle, \langle a, b \rangle, \langle b, a \rangle, \langle a, c \rangle, \langle c, a \rangle, \langle b, c \rangle, \langle c, b \rangle, \langle a, b, c \rangle, \langle a, c, b \rangle, \langle b, a, c \rangle, \langle b, c, a \rangle, \langle c, a, b \rangle, \langle c, b, a \rangle\}$. Let $|S|$ denote the cardinality of a set S , obviously $|T(Z)| > |P(Z)|$ if $|Z| > 2$.
4. The symbol $\underline{\in}$ is used to refer to an element in a tuple. For example, $a \underline{\in} \langle a, b, c \rangle$ is true.

Sets will be denoted by capital letters, whereas small letters will denote individual elements. Appendix B summarizes the set theory terms used in this chapter.

The artifact's representation scheme is primarily based on the hypothesis that any artifact is built upon the multiplicity of modules and relationships among them. We make here a fundamental assumption by letting the design representation scheme of artifacts to be discrete. This assumption is compatible with common discretization strategies characterizing the conceptual stage in the progressive refinement that occurs during a product's life-cycle.

Definition 4.2: Artifact Space (\mathcal{D}) - An artifact space is a tuple $\mathcal{D} = \langle M_0, C^0, M^* \rangle$. Alternately, \mathcal{D} is referred to as the *attribute space*, *structure space*, or the

design space. Let us discuss each component in turn,

Atomic Modules (Atomic Attributes)

M_0 is a set of atomic modules. Atomic modules cannot be defined in terms of other modules (except trivially by themselves). The set M_0 represents primitive components which can be assembled to construct complex modules. Consider for example the design of a computer environment (e.g. communication networks): The basic modules can be programs, data files, subroutines. In the design of analog circuits atomic modules may represent resistors, capacitors, operational amplifiers, and diodes.

Relations

C^0 designates a pre-assigned set of relations. $c \in C^0$ denotes a collection of tuples, each represents a relation among the elements (modules) within the tuple. The relationships among modules may also be expressed in non-mathematical terms (e.g. in logical terms). Consider for example the design of an analog circuit; the relations among atomic modules may be represented by 'information' and 'physical' relations among capacitors, resistors etc.

Complex Modules (Artifact Description)

M^* designates the set of all modules, i.e. $M^* = \{Atomic\ modules\} \cup \{Complex\ Modules\}$, where 'Complex Modules' is defined hierarchically in terms of predefined modules as follows. An artifact $m \in M^*$ is defined by a set $C \subseteq C^0$ such that $C = \{c_i\}_{i \in I}$, where $c_i = \{M_{i,k} = \langle m_{i1}, m_{i2}, \dots, m_{in} \rangle_k : c_i(\langle m_{i1}, m_{i2}, \dots, m_{in} \rangle_k)$ is true $\} \subseteq (M^*)^n$. $c_i(\langle m_{i1}, m_{i2}, \dots, m_{in} \rangle_k)$ designates a formula (a rule or behavior) associated with the relation c_i , and $(M^*)^n$ is a Cartesian product of n copies of the set M^* . The k -th element $M_{i,k}$ of c_i , is termed as the k -th assignment of the relation c_i .

Hence, an artifact m is defined in terms of assignments $\{M_{i,k}\}$ and set C ; such that $C \subseteq C^0$; and $M_{i,k}$ denotes an ordered set of modules. We define $M = \bigcup_{i,k} \{m : m \in M_{i,k}\}$ as the carrier set of m and C as the relation set of m . For convenience the notation $m = \langle M, C \rangle$ is often used.

We assume that the assignments within a relation are arranged according to a certain predefined rule (e.g., a lexicographic order rule). The set of artifacts that are

defined on the carrier set M and the relation set C is denoted by $\langle \overline{M}, C \rangle$ (thus, $m \in \langle M, C \rangle$).

Since the definition of complex modules is hierarchical, the terms 'artifact', 'complex module' and 'module' are used interchangeably. The foregoing definition allows for unfeasible artifacts. To eliminate those artifacts from further discussions, we define in the sequel legal (feasible) artifacts.

Input-Output Models

In most cases the ordered set of modules $M_{i,k}$ (an assignment of a relation c) will be included in the product set $\prod_{\alpha \in A} M_{\alpha}$ (where A is a suitably chosen index set

formalizing the different types of attributes in M^*), and hence the behavior $c \subseteq \prod_{\alpha \in A} M_{\alpha}$ becomes a relation between the basic modules $m_{\alpha} \in M_{\alpha}$, $\alpha \in A$. All further mathematical structure on product set, as maps, functions, graphs, resulting in a relation is thus in principle applicable in order to further structure mathematical design models. An important special case is recovered by assuming that $M = I \times O$, and that c is the graph of a map. In an *I/O map* it is possible to interpret the attributes of the basic modules in I as causing the attributes of basic modules in O . Note that an *I/O map* can be viewed as being described by the behavioral equation $M^O = c(M^I)$ (see Definition 4.2).

In this work, we do not make a distinction between inputs and outputs. There are a number of arguments for this point of view. First, since we have a tendency to think of mathematical models in terms of equations, most models being presented in the form, it is important to emphasize their ancillary role: it is the behavior, the solution set of the behavioral equations (that is, $c(\langle m_1, m_2, \dots, m_n \rangle)$), not the behavioral equations themselves, which is the essential result of a modeling procedure. Second, from a physical point of view, it may be unclear what the inputs and outputs are (think of the question whether an electrical circuit admits a structural or a behavioral description). Third, which modules will be used as input or output in a device may very well depend on the ultimate purpose for which it will be used (will a robot arm be programmed to follow a certain path or to exert a given force?). These and similar considerations speak in favor of taking Definition 4.2 as the starting point of a general design theory. It remains important, however, to incorporate inputs and outputs in this setting.

Types of Modules

Referring to types (categories) of modules instead of individual modules will allow for intentional characterizations; and is also needed in order to facilitate the operational definition of legal modules as given below. Let $\pi(M^*)$ be a partition of

the set of modules, i.e. a family of non-empty subsets of M^* such that each element of M^* belongs to exactly one of these subsets. Formally,

$$\pi(M^*) = \{\pi^i \mid \pi^i \in P(M^*), \pi^i \neq \emptyset, \bigcup_{i \in I} \pi^i = M^*, \text{ and } \pi^i \cap \pi^j = \emptyset \text{ for all } i, j \in I\}.$$

We will think of each π^i as a type; and assign to it a mnemonic type name \mathbf{t}^i . Every module in π^i is said to be an instance of \mathbf{t}^i , or is said to be of \mathbf{t}^i type. For example, in circuit design, modules can be divided into two mnemonic type names: $\mathbf{t}^1 = \text{'Basic-Signal'}$ (input signal) and $\mathbf{t}^2 = \text{'Compound-Signal'}$ (compound Boolean operation of several signals).

Legal Modules

One of the features of the design space is the ability to enforce legality constraints. We will characterize the term 'legality' by introducing three types of legality constraints, each represents a different aspect of legality. The first, and most general one, dictates what "configurations" - i.e., collection of assignments and their related relations - are permitted as part of legal modules. To put it in illustrative words, assume that a relation is identified with a device (or process) which impresses a specified function on quantities flowing through it (by inputting modules such as mass, energy, information, or a combination of the three). Then, the first legality constraint defines the legal relations among several devices. This is a fairly general characterization for introducing legal constraints. However, it is often useful - when constructing real design artifacts - to consider further two special cases derived from this general definition. The first special case defines the functioning of each individual relation by specifying the tuples that are legally connected via the relation. In our example, it specifies the imputed quantities of mass, energy and information flowing through each device. The second special case distinguishes the functioning of each individual module by specifying the tuples in which it is permitted to participate. In our example, it describes the combinations of flowing quantities in which each specified quantity is permitted to participate.

Function-Attribute Interface

We distinguish between the artifact space, \mathcal{A} (also called the attribute space) and the function space, \mathcal{F} (to be defined in chapter 5), each has its own algebraic structure. The relation between the artifact space and the function space is performed via the Analysis mapping, $\Gamma : \mathcal{A} \rightarrow \mathcal{F}$ which assigns to every complex module in the artifact space a functional property (or functional description) in the function space.

To clarify the above terms consider the following example. a Bondgraphs is a useful schematic language for representing designs in the function space [4]. The bondgraphs language is used for describing the exchange of energy in systems composed of lumped parameter elements. For example, transformers in bondgraphs

(part of the function space) represent elements that convert effort in one medium to effort in another medium (a functional property), and flow in one medium to flow in another medium. Examples (part of the artifact space) of transformers include a piston-cylinder that relates a fluid flowrate to velocity; and a motor that relates a torque to current or an angular velocity to voltage. These will be expressed in our formal setting as: $\Gamma\{\textit{piston-cylinder, motor}\} \rightarrow (\textit{transformer})$. The next chapter further elaborates on these issues.

4.2.2 EXAMPLES

Example 4.1: Design of a switching circuit - Consider a simplified example of a switching circuit: an electrical device having n inputs and one output. The signals are binary: T, F . In electrical terms, F could be considered as 0 potential and choose the unit of potential so that the T value has potential 1. Assume that the device has no memory; i.e., the present output level depends only on the present inputs (and not on past history). Then the output of the device is described by the Boolean function, $\mathbf{B}(X_1, X_2, \dots, X_n)$.

Devices meeting all these assumptions constitute an essential part of digital-computer circuitry. There is, for example, the two-input AND gate, for which the output is the minimum of the input. This device realizes the Boolean function $\mathbf{B}(0, 1) = \mathbf{B}(1, 0) = \mathbf{B}(0, 0) = 0$ and $\mathbf{B}(1, 1) = 1$.

The problem facing a computer engineer is: Given \mathbf{B} , find a circuit for which the cost is minimum, subject to constraints such as a maximum allowable delay. For this problem the engineer has a library of available devices. For example, the engineer might have available NOT, two-input AND, three-input OR.

An artifact, which is associated with switching circuits (from a standpoint of its wiring implementation) consists the following primitive modules and relations. The modules (M_0) of a switching circuit are of three groups:

1. The set of Input signals whose elements are denoted by A, B, C, etc.
2. The set of Available devices whose elements are denoted by AND_i , OR_i , NOT_i , XOR_i , NOR_i etc.
3. The set of Output signal whose single element is denoted by Output.

The relations (C_0) are of two groups:

1. Input relation between an input signal and a device.
2. Wiring relation between two devices.
3. Wiring relation between a device and an Output device.

For example, consider the switching circuit of Figure 4.1 to be phrased in the language of artifacts:

$$m = \{ c_1 = \{ \langle A, \text{AND}_1 \rangle, \langle A, \text{AND}_2 \rangle, \langle B, \text{AND}_2 \rangle, \langle B, \text{AND}_3 \rangle, \langle C, \text{AND}_1 \rangle, \}$$

$\langle C, \text{AND}_3 \rangle$;

$c_2 = \{ \langle \text{AND}_1, \text{OR}_1 \rangle, \langle \text{AND}_2, \text{OR}_1 \rangle, \langle \text{AND}_3, \text{OR}_2 \rangle, \langle \text{OR}_1, \text{OR}_2 \rangle \}$;

$c_3 = \{ \langle \text{OR}_2, \text{Output} \rangle \}$.

For example, $c_1 = \{ M_{11}, M_{12}, \dots, M_{15} \} = \{ \langle m_{11}, m_{12} \rangle_1, \langle m_{11}, m_{12} \rangle_2, \dots, \langle m_{11}, m_{12} \rangle_5 \}$, and $\langle m_{11}, m_{12} \rangle_1 = \langle A, \text{AND}_1 \rangle$.

It should be stressed that this representation is not unique. Consider a different setting (from a functional view) whereby modules are classified into three groups: Input, Latent and Output signals. The relations are designated by AND, OR, and NOT. Thus, for example, the above switching circuit is represented in the new setting (see Figure 4.1) as: $m = \{ \text{AND} = \{ \langle A, C, S_1 \rangle, \langle A, B, S_2 \rangle, \langle B, C, S_3 \rangle \} ; \text{OR} = \{ \langle S_1, S_2, S_4 \rangle, \langle S_4, S_3, \text{Output} \rangle \} \}$; for example, the assignment $\langle B, C, S_1 \rangle$ is interpreted as $S_1 = B \wedge C$, where ' \wedge ' represents the Boolean operator 'or'.

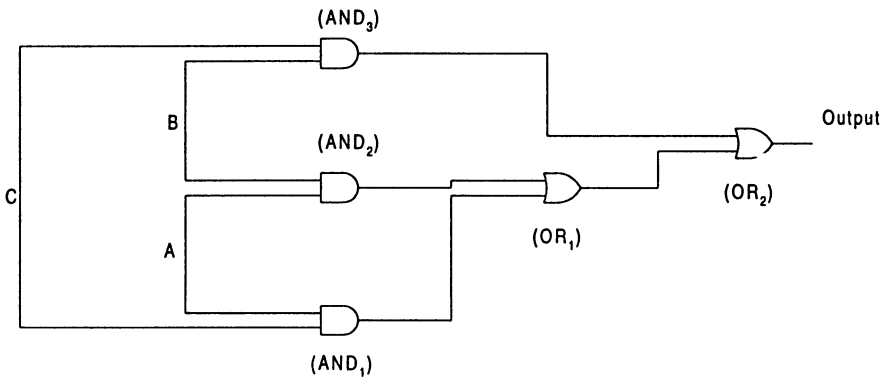


Figure 4.1 Design of A Switching Circuit

Example 4.2: Design of an electric light bulb - Consider a simplified example of the incandescent electric bulb. The basic principle underlying the electric light bulb is to raise the temperature of the filament - by transferring an electric current through it - as close as possible to its melting point, resulting in radiation of light (see Figure 4.2a-b).

An artifact, which is associated with the electric bulb (from a functional point of view) consists of the following primitive modules and relations. The modules (M_0) - classified by their physical nature - are of three groups:

Group 1: The set of Electromagnetic devices whose elements are denoted by LIGHT, CURRENT and ELECTRIC FIELD;

Group 2: The set of Mechanical devices whose elements are denoted by SUPPORTS, ROD WIRES and FILAMENT;

Group 3: The set of Flow devices whose single element is denoted by GAS.

The relations (C^0) - each expresses a functional operation which relates an input module with an output module - are of seven groups: 'emit', 'hold', 'heat', 'conduct', 'force', 'contain' and 'prevent-evaporation'.

The universal set of modules M^* is composed of four complex modules (including the primitive modules presented above):

Class of modules of order 0:

Primitive modules: LIGHT, CURRENT, ELECTRIC FIELD, SUPPORTS, ROD, WIRES, FILAMENT and GAS.

Class of modules of order 1:

GAS_BULB subsystem - $m_{11} = \{\text{contain}(\text{BULB}, \text{GAS})\}$

SUPPORT subsystem - $m_{12} = \{\text{hold}(\text{ROD}, \text{SUPPORT})\};$

CURRENT subsystem - $m_{13} = \{\text{conduct}(\text{WIRE}, \text{CURRENT}); \text{force}(\text{ELECTRIC-FIELD}, \text{CURRENT})\}$

Class of modules of order 2:

HEAT subsystem - $m_{21} = \{\text{prevent-evaporation}(m_{11}, \text{FILAMENT}); \text{hold}(m_{12}, \text{FILAMENT}); \text{heat}(m_{13}, \text{FILAMENT})\};$

Class of modules of order 3:

LIGHT-BULB system - $m_{31} = \{\text{emit}(m_{21}, \text{LIGHT})\}.$

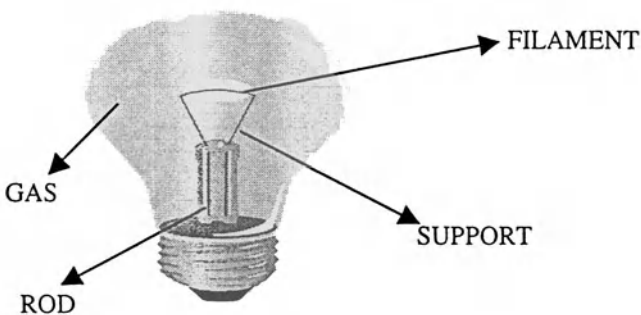


Figure 4.2a An Incandescent Bulb

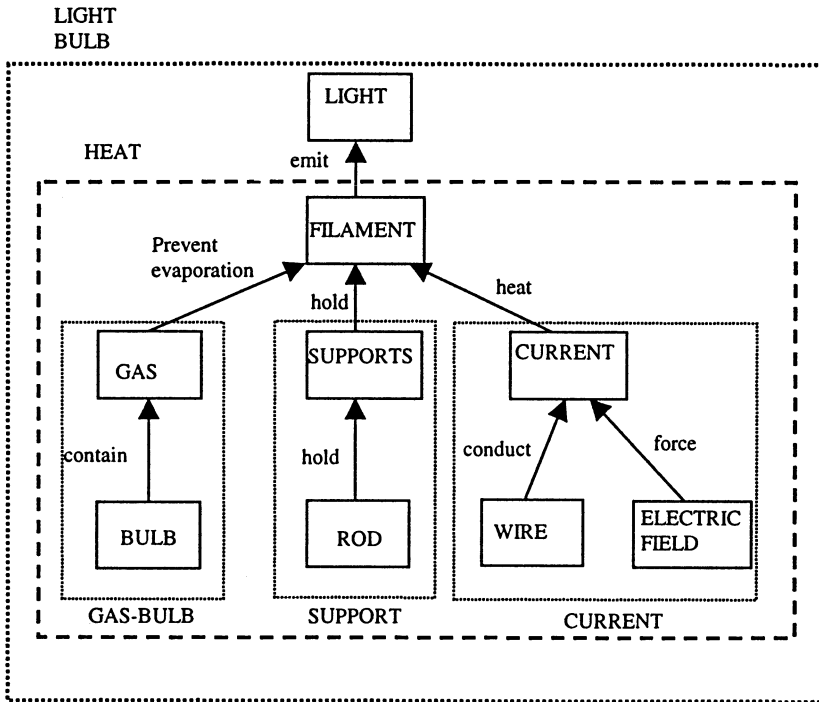


Figure 4.2 b The Light Bulb System

Example 4.3: Design of a multistorey reinforced concrete building - Figure 4.3 shows a typical floor of a three storied building that has been proposed as a design solution for a given set of client requirements.

An artifact, which is associated with the multistorey building (from a physical point of view) consists of the following primitive modules and relations: The basic modules (M_0) are represented by beams and columns. Beams and columns are considered part of a junction only if they originate from that particular junction. Therefore, junctions are considered as relations (C^0). Junctions are distinguished in accordance with the number of beams and columns that are originated from that particular junction. Consequently, we have Junction₁, Junction₂ and Junction₃ as relations. In addition, we

Number of Floors = 4

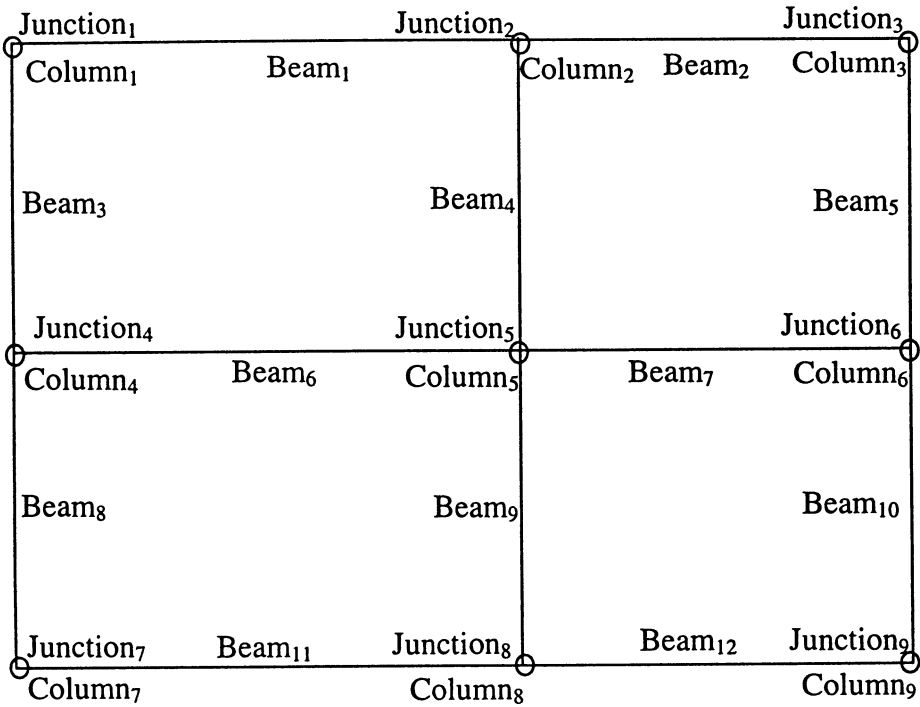


Figure 4.3 Multistorey Reinforced Concrete Building

include the relation 'Building' to indicate a collection of several junctions, thus constituting the reinforced concrete building.

The universal set of modules M^* consists the following complex modules:

Class of modules of order 0:

Primitive modules - Beam₁, Beam₂, ..., Beam₁₂, Column₁, Column₂, ..., Column₉.

Class of modules of order 1:

JUNCTION₁ subsystem - $m_{11} = \{ \text{Junction}_3(\text{Beam}_1, \text{Beam}_3, \text{Column}_1) \};$

JUNCTION₂ subsystem - $m_{12} = \{ \text{Junction}_3(\text{Beam}_2, \text{Beam}_4, \text{Column}_2) \};$

JUNCTION₃ subsystem - $m_{13} = \{ \text{Junction}_2(\text{Beam}_5, \text{Column}_3) \};$

JUNCTION₄ subsystem - $m_{14} = \{ \text{Junction}_3(\text{Beam}_6, \text{Beam}_8, \text{Column}_4) \};$

JUNCTION₅ subsystem - $m_{15} = \{ \text{Junction}_3(\text{Beam}_7, \text{Beam}_9, \text{Column}_5) \};$

JUNCTION₆ subsystem - $m_{16} = \{\text{Junction}_2(\text{Beam}_{10}, \text{Column}_6)\};$

JUNCTION₇ subsystem - $m_{17} = \{\text{Junction}_2(\text{Beam}_{11}, \text{Column}_7)\};$

JUNCTION₈ subsystem - $m_{18} = \{\text{Junction}_2(\text{Beam}_{12}, \text{Column}_8)\};$

JUNCTION₉ subsystem - $m_{19} = \{\text{Junction}_1(\text{Column}_9)\};$

Class of modules of order 2:

BUILDING FLOOR system - $m_{21} = \{\text{Building}(m_{11}, m_{12}, \dots, m_{19})\}.$

Example 4.4 (piston subassembly): That any knowledge representation is built upon the multiplicity of objects and relationships among them is an empirical proposition for which evidence has to be provided. Here we provide further evidence corroborating the proposition by demonstrating how traditional computer models of geometrically complex objects conform to the entity-relational knowledge representation hypothesis.

The problem of building computer models of geometrically complex objects has been addressed in the context of graphics [2], computer aided design [5] and mechanical assembly [3]. Two major methods have been devised. The *surface* approach [1] describes the surfaces of the objects by specifying the vertices and edges of the faces of polyhedra or, for curved objects, the crosssections or surface patches. The *solid* approach [2] approximates complex objects by composing several simpler volumes. There also exist some hybrid systems that allow both types of descriptions.

The Artifact Description

Figure 4.4 shows a piston subassembly from a model aircraft engine. Figure 4.5 shows a schematic description of the parts in the piston component subassembly. The parts are arranged hierarchically, where any desired subparts can be represented as nodes in the part model trees.

Each node has information regarding the *size*, *type* and *relative position* of the subparts. All the subparts, including holes, are approximated as *rectangular* or *octagonal right prisms*. This provides a uniform internal representation for all the object types. This representation simplifies the definitions of the spatial modeling operations. By generalizing to polyhedra we could approximate the desired volumes to any required accuracy.

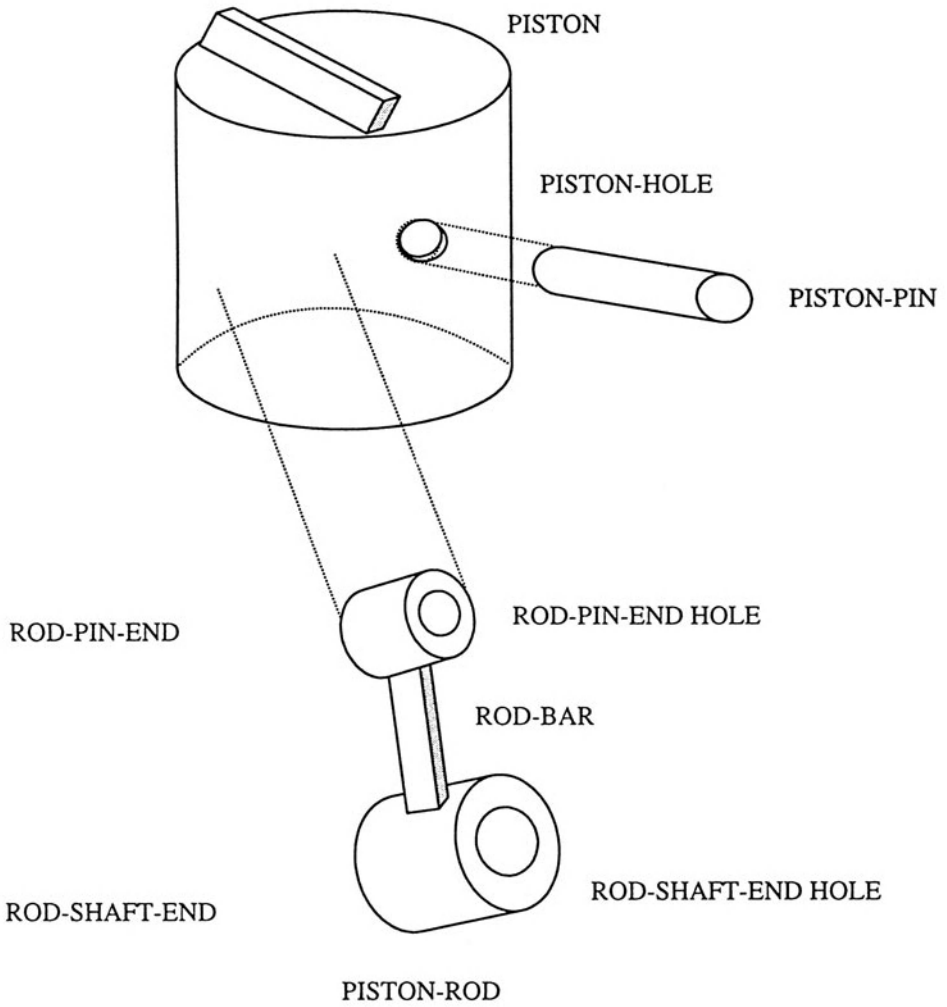


Figure 4.4 The Piston Subassembly of A Model Aircraft Engine

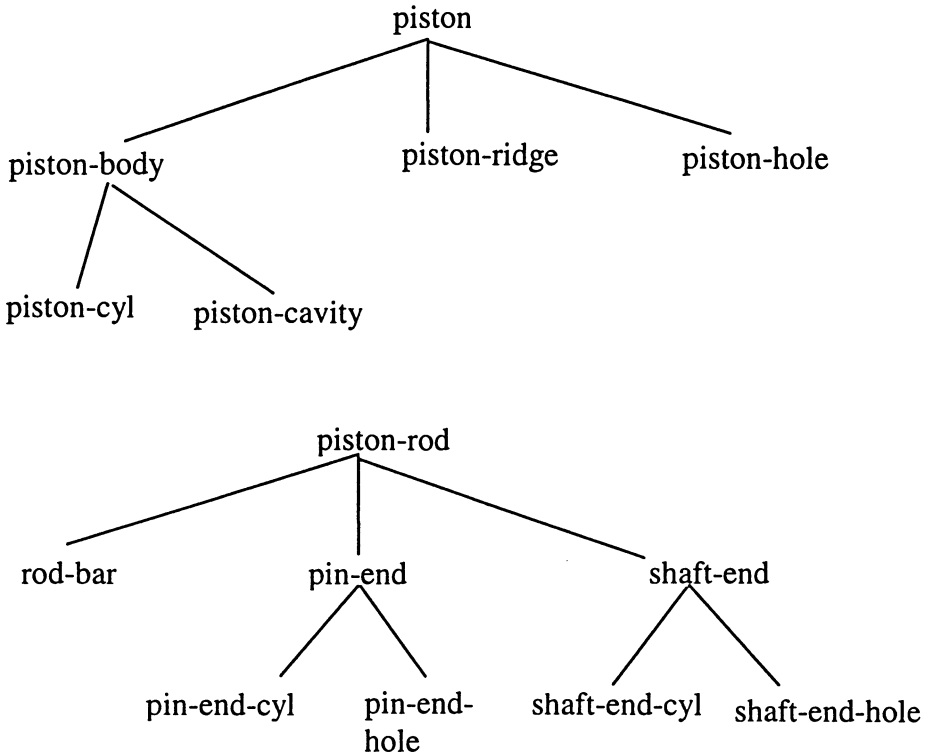


Figure 4.5 The Tree-structured Relationship of Parts in the Piston Subassembly

A Formal Representation Scheme

Primitive modules

- Polyhedral solids whose crosssections are regular polygons. For example: (1) a rectangular solid, and (2) an octagonal solid which is meant to approximate a cylinder.
- Properties, attributed to primitive objects, that specify their size parameters, vertex points, equations for the planes of the faces, generalized position and, orientation, etc. For example, 'LENGTH', ' and 'RADIUS,' are considered as basic modules.
- Additional primitive modules include: 'TYPE' which denotes the polyhedral solid that approximates the object; and 'X', 'Y' and 'Z' which denote the position of the object.

Complex Modules

Complex modules are represented as unions of other objects (primitive as well as complex). Holes and cut-outs can be treated uniformly as objects by allowing primitive objects to have negative volumes. For example, the cavity of the piston, shown in Figure 4.4, can be represented as two cylindrical holes and a cuboid to approximate its elliptical cross-section.

Relations

The set of relations includes: 'offset,' 'angles,' and 'link' which indicate the coordinate transformation between the local coordinate systems of two objects; the equality relation '='; and the relations 'bar,' 'shaft-end-cyl,' 'shaft-end-hole,' 'pin-end-cyl,' and 'pin-end-hole'.

Modeling the Piston Rod

Let us present the model of the piston rod for the model aircraft engine (see Figure 4.4). First we define the components parts of the object. The components parts are: ROD-BAR, PIN-END, SHAFT-END, PIN-END-CYL, PIN-END-HOLE, SHAFT-END-CYL, and SHAFT-END HOLE. Formally:

BAR: bar(< TYPE = RECT, X = 0.2, Y = 0.2, Z = 0.62 >)

SHAFT-END-CYL: shaft-end-cyl(<TYPE = CYL, RADIUS = 0.156, LENGTH = 0.114 >)

SHAFT-END-HOLE: shaft-end-hole(<TYPE = CYL-HOLE, RADIUS = 0.089, LENGTH = 0.114 >)

PIN-END-CYL: pin-end-cyl(< TYPE = CYL, RADIUS = 0.134, LENGTH = 0.16 >)

PIN-END-HOLE: pin-end-hole (<TYPE = CYL-HOLE, RADIUS = 0.081, LENGTH = 0.16 >)

Note that the foregoing objects define complex modules of order 2. For example, 'RECT' is a primitive module, 'TYPE = RECT' is complex module of order 1, and BAR is a complex module of order 2.

The next step is to indicate the relationships between the various parts. The simplest links in the model of the piston rod are the relationships of the holes to their corresponding cylinders since they are aligned and concentric:

link (<SHAFT-END-HOLE, SHAFT-END-CYL>)

link (<PIN-END-HOLE, PIN-END-CYL>)

Then the hollow cylinders are placed at the ends of the bar:

link (<SHAFT-END-CYL, BAR>)

link (<PIN-END-CYL, BAR>)

Finally, we need also to denote the position and orientation of either one of the hollow cylinders relative to the BAR:

offset (<SHAFT-END-CYL, BAR, X = 0, Y = 0, Z = 0.466>)
 angles (<SHAFT-END-CYL, BAR, X = 0, Y = p/2, Z = 0>)
 offset(<PIN-END-CYL, BAR, X = 0, Y = 0, Z = -0.444>)
 angles (<PIN-END-CYL, BAR, X = 0, Y = p/2, Z = 0>)

Example 4.5: Representing artifact in propositional calculus - In Chapter 5 (Definition 5.5), we make several simplifying assumptions about the nature of artifact representation. The description of artifacts is limited to a list of properties, where an artifact may or may not satisfy a particular property. To describe how properties are constructed we need the following:

1. a finite set of names for basic properties (the basic modules). We choose to use m_1, m_2, \dots, m_k . These basic property letters are intended to be interpreted as the names for concrete properties such as *high strength*, or *actuation ease*.
2. The relationships between various properties can be represented symbolically, using *connectors* (the relations). A connector is a function that makes a compound properties out of simple properties. The following connectors are defined as a matter of convention: (1) the conjunction connector \wedge , which is intended to be read as “and,”; (2) the disjunction connector \vee , which is intended to be read as “or”; and (3) the negation connector \sim , which is intended to be read as “not,”. The foregoing connectors can be used to combine simple properties letters, as in $(\sim m_2 \wedge (m_4 \vee m_1))$. We use A, B, C and A_1, A_2, \dots to stand for complex properties of artifacts (the complex modules).
3. The set of all artifact descriptions (the attribute space) is the set of *all* properties obtained by conjunction (\wedge), disjunction (\vee) and negation (\sim) over the basic properties.

Given properties A and B, the significance of the connectors are the following:

1. $\sim A$ is satisfied by the artifact if A is not satisfied by the artifact, or conversely, unsatisfied if A is satisfied.
2. $A \wedge B$ is satisfied if both referent statements A and B are satisfied. If one or both of A and B are unsatisfied, then the compound property is unsatisfied.
3. $A \vee B$ is satisfied if A or B is satisfied. The compound property is unsatisfied only when both A and B are unsatisfied.

4.2.3 PROPERTIES OF THE DESIGN SPACE

Definition 4.3: The Graph Incurred by \mathcal{D} - A directed graph $G_{M^*} = (V, E)$ is referred to as the directed graph which incurred by \mathcal{D} if $V = M^*$ & $\langle m_1, m_2 \rangle \in E$

$$\Leftrightarrow \exists M \neq \emptyset, \exists C \neq \emptyset : (m_1 \in M \ \& \ m_2 \in \overline{\langle M, C \rangle}).$$

Definition 4.4: Regular Artifact Space - An artifact space \mathcal{D} whose incurred graph G_{M^*} is a directed acyclic graph is termed as a regular artifact space. For brevity 'modules' will be used, when the context identifies 'complex modules' from 'atomic modules'.

Remark 4.1: The rationale which guides the above definition is that we do not allow a module m_1 to be used in defining itself. Informally, if G_{M^*} is not acyclic then there exists a walk $\langle m_1, m_2 \rangle \langle m_2, m_3 \rangle \dots \langle m_{k-1}, m_1 \rangle$ that joins m_1 to itself, which means that $\forall i: m_i$ is used in defining m_{i+1} . This situation is unrealistic in most engineering design applications.

Remark 4.2: It can be easily verified, applying Definition 4.4, that any regular artifact space \mathcal{D} can be arranged in classes as follows,

- Class of modules of order 0: modules which belong to the set M_0 (Atomic modules).
- Class of modules of order i : modules which belong to the set $M_i = \{m_{ij}\}_j$ where $m_{ij} \in \overline{\langle \tilde{M}_{ij}, C_{\tilde{M}_{ij}} \rangle}$; $C_{\tilde{M}_{ij}}$ is the relation set, and \tilde{M}_{ij} is the carrier set that satisfies $\tilde{M}_{ij} \subseteq \bigcup_{k=0}^{i-1} M_k$. M^* is given formally by $M^* = \sup_j \{M_j\}$.

Remark 4.3: We focus in this book only on regular artifact spaces, which graphically correspond to acyclic directed graphs. In practice we have many situations that presuppose acyclicity among modules. For example, most object modeling methods - in the context of computer aided design, graphics and mechanical assembly - for representing geometrically complex objects, are implemented as regular artifact spaces.

Remark 4.4: A module m denotes an equivalence class of tuples. The notation m represents the idea that a module can be expressed in different ways, i.e. if a module m can be expressed by $\langle M^1, C_{M^1} \rangle$ or $\langle M^2, C_{M^2} \rangle$ then it is not necessarily implies that either $M^1 = M^2$ or $C_{M^1} = C_{M^2}$ (where M^i and C_{M^i} are sets; $i = 1, 2$).

The next definition allows connecting a module to itself by applying a unit relation:

Definition 4.5: Unit relation - A relation $c_1 \in C^0$ such that, $\forall M \subseteq M^* : \langle M, C \rangle = M \Leftrightarrow C = \{c_1\}$.

It is convenient, for later use, to denote the Order of a set of modules.

Definition 4.6: Modules' Order - Let M^* be a regular artifact space, then define the mapping $Order : 2^{M^*} \rightarrow \mathbb{N}$ as:

1. If $m \in M^* : Order(m) = k$ iff $m \in M_k$ and $\forall i < k : m \notin M_i$.
2. If $M = \{m_j\}_j$ then $Order(M) = \max_{m_j \in M} \{Order(m_j)\}$.

Remark 4.5: A module m can be included in several classes. Definition 4.6 (1) defines the module's order, k , as the order of the first class, M_k , which the module is included in. An alternative definition would be to replace, in the above, the 'first class' with the 'last class'. Formally: If $m \in M^* : Order(m) = k$ iff $m \in M_k$ and $\forall i > k : m \notin M_i$ (when needed, this is referred to as *Maxorder* operator).

Remark 4.6: The Order mapping may indicate the extent of 'structural complexity' inherent in a set of modules.

Definition 4.7: Strictly Regular Artifact Space - An artifact space \mathcal{O} is termed strictly regular if the following holds: $m = \langle M, C \rangle$ and $M \neq \{m\} \Rightarrow m \notin M$ and $Order(m) > Order(M)$.

Remark 4.7: Definition 4.7 implies that $m \in M_0 \Rightarrow [m = \langle M, C \rangle \Rightarrow M = \{m\}]$ and $C = \{c_1\}$.

Definition 4.8: Predicate - Let P denotes a predicate (e.g., a unary predicate) that is assigned to individuals modules or relations.

Remark 4.8: Consider for example the predicate, which is assigned to modules, of 'being of order k '.

The remainder of this section elaborates on properties of the design space model, by inducing on it three types of operators. These operators enable to perform forward search (from initial modules toward modules of higher order) and backward search (from complex modules toward modules of lower order). Moreover, it is shown that it is sufficient to use these operators in order to perform any search tasks over a given design space.

Definition 4.9: Composition Operator - A mapping $\Psi_S^{(k)} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$.

Let $(M, C) \in \text{Domain}(\Psi_S^{(k)})$ then $\Psi_S^{(k)}(M, C)$ is defined in terms of a positive integer k , non-empty sets of indices I_1, I_2, \dots, I_k and relations $\{C^{l,i}\}$ for every $1 \leq l \leq k, i \in I_l$ where $C^{l,i} \subseteq C$, such that:

1. $M_0 = M$.
2. For every $1 \leq l \leq k : M^l \subseteq \overline{\bigcup_{i \in I_l} \langle M^{l,i}, C^{l,i} \rangle}$ such that $M^{l,i} \subseteq M^{l-1}$ for every $i \in I_l$ and $\neg(\exists i \in I_l, \forall m \in M^{l-1} : m \notin \overline{\langle M^{l,i}, C^{l,i} \rangle})$.
3. $\Psi_S^{(k)}(M, C) = M^k$

Remark 4.9: When $k = 1$, a Composition operator is termed First-Order Composition operator and is denoted by $\Psi_S^{(1)}$. When $k > 1$, a Composition operator is termed k -order Composition operator and is denoted by $\Psi_S^{(k)}$. For brevity, the order of a Composition operator is omitted when needed. For example, related to our circuit example we define the following First-Order Composition operator $\Psi_S^{(1)}(\{A, B, C, \text{AND}_1, \text{AND}_2, \text{OR}_1, \text{OR}_2, \text{OUTPUT}\}, \{c_1, c_2, c_3\}) = \{m\}$, where m denotes the artifact description of the circuit in Figure 4.1.

Observation 4.1: Decomposing $\Psi_S^{(k)} - \Psi_S^{(k)}$ may be decomposed in form $\Psi_S^{(k)} = \Psi_{S,k}^{(1)} \circ \Psi_{S,k-1}^{(1)} \circ \dots \circ \Psi_{S,1}^{(1)}$ (we define $(\Psi_{S,i} \circ \Psi_{S,j})(M, C) \equiv \Psi_{S,i}(\Psi_{S,j}(M, C), C)$). $\Psi_{S,i}^{(1)}$ are First-Order Composition operators and $k \geq 1$.

Remark 4.10.1: Observation 4.1 is important since it enables the designer to restrict himself to basic definitions of First-Order Composition operators; thus constructing k -Order operators by composing First-Order operators. It also facilitates the forthcoming analysis.

Remark 4.10.2: Further conditions may be imposed on Composition operators; i.e., $\Psi_S^{(k)}$ may satisfy:

1. $\Psi_S^{(k)}(M^1 \cup M^2, C) \supseteq \Psi_S^{(k)}(M^1, C) \cup \Psi_S^{(k)}(M^2, C)$ (Semi-Additivity)
2. $M^1 \subseteq M^2 \Rightarrow \Psi_S^{(k)}(M^1, C) \subseteq \Psi_S^{(k)}(M^2, C)$ (Monotonicity)
3. $M^1 \subseteq \Psi_S^{(k)}(M^1, C)$ (Inclusion)
4. $\Psi_S^{(k)}(M^1, C) \subseteq \Psi_S^{(1)}(\Psi_S^{(k-1)}(M^1, C), C)$ (First-Order Inclusion)

Condition 1 asserts that the set of modules that can be generated (by Composition) by joining two sets of modules is at least equal to the set of modules that is the join of the generated sets, obtained from each set separately. Condition 2 and Condition 4 are resulted from Condition 1. Condition 3 implies the following property: $\forall(M, C) \in \text{Domain}(\Psi_S^{(k)}), \bar{M} = \Psi_S^{(k)}(M, C): (\text{Order}(M) > \text{Order}(C))$ i.e., the set of modules M constructed by the Composition operator are of higher order than the order of C - the set on which it acts upon - for every $C \subseteq C^0$.

Definition 4.10: Proper Composition Operator - A Composition operator, denoted by Ψ_{SP} , which satisfies the following conditions:

1. $\exists(M, C) \in \text{Domain}(\Psi_{SP}): \Psi_{SP}(M, C) = M$ (Nonunity)
2. $(\forall(M, C) \in \text{Domain}(\Psi_{SP}): \Psi_{SP}(M, C) = M_\emptyset) \Rightarrow M = M_\emptyset$ (Regularity)

Remark 4.11: Nonunity means that new modules are obtained by a Proper Composition operator for some tuple (M, C) ; while Regularity means that a Composition operator, Ψ_S , generates legal tuples for every $(M, C) \in \text{Domain}(\Psi_S)$ and $M \neq M_\emptyset$.

Definition 4.11: Projection Operator (Relative to predicate Π) - A mapping $\Psi_P^\Pi: \Omega \rightarrow \Omega$, defined on a set Ω , such that: $\Psi_P^\Pi(\omega) = \omega$ if $\Pi(\omega)$ is True and \emptyset otherwise.

A Projection operator partitions the set Ω into two disjoint sets, Ω^F and Ω^T , where Ω^T is defined as: $\Omega^T = \{\omega: \Pi(\omega) \text{ is True}\}$. We let $\Omega^F = \Omega - \Omega^T$.

The following demonstrates a special type of a Projection Operator.

Definition 4.12: Indicator Operator - A Projection Operator, defined on 2^{M^*} (2^{C^0} , respectively) such that:

$$\Psi_P^{R(M)}(\bar{M}) = \begin{cases} M & \text{if } \bar{M} = M \\ M_\emptyset & \text{otherwise} \end{cases} \quad \text{or} \quad \Psi_P^{R(C)}(\bar{C}) = \begin{cases} C & \text{if } \bar{C} = C \\ C_\emptyset & \text{otherwise} \end{cases}$$

Remark 4.12: In terms of the above terminology, the Property $\Pi = R(M)$ is interpreted as 'Equality to modules set M '.

Lemma 4.1: Decomposition Lemma - Let Ψ_S be a Composition operator that satisfies the following conditions:

1. $\exists(M, C) \in \text{Domain}(\Psi_S)$ such that $\Psi_S(M, C) \neq M_\emptyset$ (Feasibility)
2. $\exists(M, C) \in \text{Domain}(\Psi_S)$ such that $\Psi_S(M, C) \neq M$ (Nonunity)

Then Ψ_S may be decomposed in the form $\Psi_{SP} \circ \Psi_P^\Pi$, i.e. $\Psi_S(M, C) = \Psi_{SP}(\Psi_P^\Pi(M, C))$, where Ψ_{SP} is a Proper Composition operator and Ψ_P^Π is a Projection operator over the product set of modules and relations.

Definition 4.13: First-Order Closure Composition Operator - A mapping $\Psi_{\bar{S}}^{(1)} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$ that satisfies the following condition: Let $(M, C) \in \text{Domain}(\Psi_{\bar{S}}^{(1)})$. Define a Closure First-Order Composition operator as: $\Psi_{\bar{S}}^{(1)}(M, C) = \text{sup}\{M^i \text{ such that } \exists \Psi_{\bar{S}}^{(1)} : \Psi_{\bar{S}}^{(1)}(M, C) = M^i\} = \overline{\bigcup_{M^j \subseteq M, C^j \subseteq C} \langle M^j, C^j \rangle}$. The latter equality is easily inferred by Definition 4.9.

Remark 4.13.1: $\Psi_{\bar{S}}^{(1)}(M, C)$ specifies the set of all modules, which is generated in a single composition step.

Remark 4.13.2: Denote a k -Order Closure Composition operator to be the composition of k First-Order Closure Composition operators i.e. $\Psi_{\bar{S}}^{(k)} = \Psi_{\bar{S}}^{(1)} \circ \Psi_{\bar{S}}^{(1)} \circ \dots \circ \Psi_{\bar{S}}^{(1)}$.

Definition 4.14: Closure Composition Operator - A mapping $\Psi_{\bar{S}} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$ such that $\forall(M, C) \in \text{Domain}(\Psi_{\bar{S}}) : \Psi_{\bar{S}}(M, C) = \bigcup_{k=1}^{\infty} \Psi_{\bar{S}}^{(k)}(M, C)$

Remark 4.14: $\Psi_{\bar{S}}(M, C)$ specifies the set of all modules, \overline{M} , which can be generated from a pair (M, C) by composition. It is easily verified, using the compositional structure of complex modules (Remark 4.2), that for regular artifact spaces: $\Psi_{\bar{S}}(M_0, C^0) = M^*$.

Lemma 4.2: Let $\Psi_{\bar{S}}^{(k)}(\cdot)$ be k -Order Closure Composition operator. Assume $c_1 \in C$ then the following properties hold:

1. $\Psi_{\bar{S}}^{(k)} \left(\bigcup_{i=1}^N M^i, C \right) \supseteq \bigcup_{i=1}^N \Psi_{\bar{S}}^{(k)} (M^i, C)$ (Semi-Additivity)
2. $M^1 \subseteq M^2 \Rightarrow \Psi_{\bar{S}}^{(k)} (M^1, C) \subseteq \Psi_{\bar{S}}^{(k)} (M^2, C)$ (Monotonicity)
3. $M^1 \subseteq \Psi_{\bar{S}}^{(k)} (M^1, C)$ (Inclusion)

Corollary 4.1: It is easily inferred, by applying Property 3 that, $k_1 < k_2 \Rightarrow \Psi_{\bar{S}}^{(k_1)} (M^1, C) \subseteq \Psi_{\bar{S}}^{(k_2)} (M^1, C)$, for every k_1, k_2 .

Corollary 4.2: The Closure Composition operator satisfies the following property: $\Psi_{\bar{S}}^{(k)} \left(\bigcap_{i=1}^N M^i, C \right) \subseteq \bigcap_{i=1}^N \Psi_{\bar{S}}^{(k)} (M^i, C)$

Definition 4.15: Decomposition Operator - A mapping $\Psi_A^{(k)} : 2^{M^*} \rightarrow 2^{M^*}$. Let $M \in \text{Domain}(\Psi_A^{(k)})$, then $\Psi_A^{(k)}$ is defined in terms of a positive integer k , nonempty sets of indices I_1, I_2, \dots, I_k and relations $\{C^{l,i}\}$ for every $1 \leq l \leq k$ and $i \in I_l$, such that:

1. $M^0 = M$
2. For every $0 \leq l \leq k - 1$: $M^l \subseteq \overline{\bigcup_{i \in I_l} \langle M^{l,i}, C^{l,i} \rangle}$ such that $\bigcup_{i \in I_l} M^{l,i} = M^{l+1}$ and $\neg(\exists i \in I_l, \forall m \in M^l : m \notin \langle M^{l,i}, C^{l,i} \rangle)$
3. $\Psi_A^{(k)} (M) = M^k$

Remark 4.15.1: Definition 4.15 implies that any Decomposition operator has a 'companion' operator (not unique), which is a Composition operator denoted by $\tilde{\Psi}_A$. $\tilde{\Psi}_A$ satisfies, beside the conditions posed in Definition 4.9, the condition:

$\bigcup_{i \in I_l} M_{l,i} = M^{l-1}$ for every $1 \leq l \leq k$. Formally, let $\bar{M} = \Psi_A^{(k)} (M)$, then $\exists C \subseteq C^0$:

$$\tilde{\Psi}_A^{(k)} (\bar{M}, C) = M.$$

Remark 4.15.2: Let \mathcal{D} be a regular artifact space, then the following is satisfied: $\forall M \in \text{Domain}(\Psi_A^{(k)})$, $\bar{M} = \Psi_A^{(k)} (M)$: $\{\text{Order}(M) \leq \text{Order}(\bar{M})\}$ i.e., the modules constructed by a Decomposition operator are of lower order than the order of the modules set, on which it acts upon.

Remark 4.15.3: When $k=1$ in Definition 4.15, a Decomposition operator is termed First-Order Decomposition operator, denoted by $\Psi_A^{(1)}$. When $k > 1$, a Decomposition operator is termed k -Order Decomposition operator, denoted by $\Psi_A^{(k)}$. For example, related to our switching circuit example, we define the following First-Order Decomposition operator $\Psi_A^{(1)}(m) = \{A, B, C, \text{AND}_1, \text{AND}_2, \text{OR}_1, \text{OR}_2, \text{OUTPUT}\}$, where m denotes the artifact description of the circuit in Figure 4.1.

Observation 4.2: Decomposing $\Psi_A^{(k)}$ - Let $\Psi_A^{(k)}$ be a Decomposition operator over 2^{M^*} . Then $\Psi_A^{(k)}$ can be decomposed in form $\Psi_A^{(k)} = \Psi_{A,k}^{(1)} \circ \Psi_{A,k-1}^{(1)} \circ \dots \circ \Psi_{A,1}^{(1)}$ (we define $(\Psi_{A,i} \circ \Psi_{A,j})(M) = \Psi_{A,i}(\Psi_{A,j}(M))$). $\Psi_{S,i}^{(1)}$ are First-Order Decomposition operators and $k \geq 1$.

Remark 4.16: Further conditions may be imposed on Decomposition operators. For example:

1. $\Psi_A^{(k)}(M^1 \cup M^2) = \Psi_A^{(k)}(M^1) \cup \Psi_A^{(k)}(M^2)$ (Additivity)
2. $M^1 \subseteq M^2 \Rightarrow \Psi_A^{(k)}(M^1) \subseteq \Psi_A^{(k)}(M^2)$ (Monotonicity)
3. $M^1 \subseteq \Psi_A^{(k)}(M^1)$ (Inclusion)
4. $\Psi_A^{(k)}(M^1) \subseteq \Psi_A^{(1)}(\Psi_A^{(k)}(M^1))$ (First-Order Inclusion)

Condition 1 asserts the Additivity of Decomposition operators, whereas Conditions 2-4 are interpreted as in Remark 4.10.2.

Definition 4.16: First-Order Closure Decomposition operator - A mapping $\Psi_{\bar{A}}^{(1)} : 2^{M^*} \rightarrow 2^{M^*}$, which is defined as follows: Let $M \in \text{Domain}(\Psi_{\bar{A}}^{(1)})$ then, $\Psi_{\bar{A}}^{(1)}(M) = \text{sup}\{M^i \text{ such that } \exists \Psi_A^{(1)} : \Psi_A^{(1)}(M^i) = M\}$.

Remark 4.17: Denote a k -Order Closure Decomposition operator to be the composition of k First-Order Closure Composition operators, formally $\Psi_{\bar{A}}^{(k)} = \Psi_{\bar{A}}^{(1)} \circ \Psi_{\bar{A}}^{(1)} \circ \dots \circ \Psi_{\bar{A}}^{(1)}$.

The following definition is needed in proving Lemma 4.3 below:

Definition 4.17: Restriction of A First-Order Composition operator - Let $\Psi_S^{(1)}$

be First-Order Composition operator, and consider $\Psi_S^{(1)}(M, C) = M^1 \cup M^2$. Recall that Definition 4.9 implies $\Psi_S^{(1)}$ to satisfy $M^1 \cup M^2 \subseteq \overline{\bigcup_{i \in I_1} \langle M^{1,i}, C^{1,i} \rangle}$, where $M^{1,i} \subseteq M$ for every $i \in I_1$. Let $I_{1,1} \cup I_{1,2} = I_1$, such that $M^1 \subseteq \overline{\bigcup_{i \in I_{1,1}} \langle M^{1,i}, C^{1,i} \rangle}$ and $M^2 \subseteq \overline{\bigcup_{i \in I_{1,2}} \langle M^{1,i}, C^{1,i} \rangle}$. Let the restriction of $\Psi_S^{(1)}$, relative to the sets M^1 and M^2 , to be the First-Order Composition operators $\Psi_{S,M^1}^{(1)}$ and $\Psi_{S,M^2}^{(1)}$ which are defined as follows:

$$\begin{aligned} \tilde{M}^1 &= \bigcup_{i \in I_{1,1}} M_{1,i} \Rightarrow \Psi_{S,M^1}^{(1)}(\tilde{M}^1, C) = M^1 \text{ and } \tilde{M}^2 = \bigcup_{i \in I_{1,2}} M_{1,i} \Rightarrow \\ \Psi_{S,M^2}^{(1)}(\tilde{M}^2, C) &= M^2. \end{aligned}$$

Lemma 4.3: $\Psi_{\tilde{A}}^{(k)}$ satisfy the following properties:

1. $\Psi_{\tilde{A}}^{(k)}\left(\bigcup_{i=1}^N M^i\right) = \bigcup_{i=1}^N \Psi_{\tilde{A}}^{(k)}(M^i)$ (Additivity)
2. $M^1 \subseteq M^2 \Rightarrow \Psi_{\tilde{A}}^{(k)}(M^1) \subseteq \Psi_{\tilde{A}}^{(k)}(M^2)$ (Monotonicity)
3. $M^1 \subseteq \Psi_{\tilde{A}}^{(k)}(M^1)$ (Inclusion)

Corollary 4.3: It is easy to show (by applying Property 3) that

$$k_1 \leq k_2 \Rightarrow \Psi_{\tilde{A}}^{(k_1)}(M) \subseteq \Psi_{\tilde{A}}^{(k_2)}(M) \text{ for every } k_1, k_2.$$

Corollary 4.4: Closure Decomposition operators also satisfy the following properties:

$$\Psi_{\tilde{A}}^{(k)}\left(\bigcap_{i=1}^N M^i\right) \subseteq \bigcap_{i=1}^N \Psi_{\tilde{A}}^{(k)}(M^i)$$

Definition 4.18: Closure Decomposition Operator - A mapping $\Psi_{\tilde{A}} : 2^{M^*} \rightarrow 2^{M^*}$, that satisfies the following condition: Let $M \in \text{Domain}(\Psi_{\tilde{A}})$, then $\Psi_{\tilde{A}}(M) = \bigcup_{k=1}^{\infty} \Psi_{\tilde{A}}^{(k)}(M)$.

Remark 4.18: A Closure Decomposition Operator generates the set of all modules, which can be synthesized to obtain its origin set M .

Observation 4.3: Let \mathcal{D} be a strictly regular artifact space, and let $\Gamma_1 = \{\Psi_{A,i}^{(1)}\}_i$ be an infinite collection of First-Order Decomposition operators. Define the collection $\{\Psi_A^{(k)}\}_k$ such that,
 $\Psi_A^{(k)} = \Psi_{A,k}^{(1)} \circ \Psi_{A,k-1}^{(1)} \circ \dots \circ \Psi_{A,1}^{(1)}$. Assume $\forall M \subset M_0, \forall \Psi_{A,i}^{(1)}: \Psi_{A,i}^{(1)}(M) \neq M$.
 Then, there exists a finite integer $k^*(M)$ such that, $\Psi_A^{(k^*(M))}(M) = \{\bar{M} : \bar{M} \subseteq M_0\}$.

The next result shows that our list of design-space operators is complete, in the sense that every legal state of the design space can be transformed into another legal state using only these operators.

Lemma 4.4: Let $m = \langle M, C \rangle$ be a complex module of the regular artifact space \mathcal{D} . Then m can be obtained from the class of basic modules, M_0 , by a finite composition of first-order composition operators.

Definition 4.19: k_1 - k_2 Order Integration Operator - A mapping $\Psi_C^{(k_1, k_2)} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$, which is defined in terms of Decomposition and Composition operators as follows:
 $\exists \Psi_A^{(k_2)}, \exists \Psi_S^{(k_1)} : \Psi_C^{(k_1, k_2)} = \Psi_S^{(k_1)} \circ \Psi_A^{(k_2)}$ for some k_1, k_2

Remark 4.19: Definition 4.19 suggests a general method, whereby subsets of the design space can be constructed by means of basic Decomposition and composition operations.

Definition 4.20: k_1 - k_2 Closure Integration Operator - A mapping, $\Psi_{\bar{C}}^{(k_1, k_2)} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$, which is defined as follows: Let $(M, C) \in \text{Domain}(\Psi_{\bar{C}}^{(k_1, k_2)})$, then $\Psi_{\bar{C}}^{(k_1, k_2)}(M, C) = \sup\{M^i \text{ such that } \exists \Psi_C^{(k_1, k_2)} : \Psi_C^{(k_1, k_2)}(M, C) = M^i\}$.

We conclude the presentation of design space operators, by introducing Closure Integration operators:

Definition 4.21: Closure Integration operator - A mapping $\Psi_{\bar{C}} : 2^{M^*} \times 2^{C^0} \rightarrow 2^{M^*}$, such that $\Psi_{\bar{C}}(M, C) = \bigcup_{k_1, k_2} \Psi_{\bar{C}}^{(k_1, k_2)}$.

Remark 4.20: A Closure Integration operator, over (M, C) , represents the set of

all modules that can be constructed from the tuple.

Observation 4.4: The following are satisfied:

1. Let $(M, C) \in \text{Domain}(\Psi_{\bar{C}}^{(k_1, k_2)})$ and $c_1 \in C$ then, $\Psi_{\bar{C}}^{(k_1, k_2)} = \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)}$
2. $\forall k_1: k_2 \leq \tilde{k}_2 \Leftrightarrow \Psi_{\bar{C}}^{(k_1, k_2)}(M, C) \subseteq \Psi_{\bar{C}}^{(k_1, \tilde{k}_2)}(M, C)$
3. $\forall k_2: k_1 \leq \bar{k}_1 \Leftrightarrow \Psi_{\bar{C}}^{(k_1, k_2)}(M, C) \subseteq \Psi_{\bar{C}}^{(\bar{k}_1, k_2)}(M, C)$
4. $\Psi_{\bar{C}}(M, C) \subseteq \Psi_{\bar{S}}(\Psi_{\bar{A}}(M), C)$

4.3 SUMMARY

To summarize, this chapter has provided guidelines and tools for the representation of design knowledge. The assumptions used to define the artifact representation scheme were highlighted and analyzed.

This chapter has not addressed the issue of how to generate efficient, topological or even hierarchical knowledge structures that support extensional as well as intentional descriptions. Thus, the scope of the artifact representation scheme is broadened in the next chapter, where an intensional representation of artifacts via *propositional calculus* (which complies with the entity-relational knowledge representation as presented in this chapter) is described. Furthermore, the intuitive concept of design process as a mapping from the desired function and constraints, called specification, to the artifact description is formalized in the next chapter by introducing the notion of *idealized design process*.

APPENDIX A (PROOFS)

PROOF OF OBSERVATION 4.1

Define $\Psi_{S,l}^{(1)}$, $1 \leq l \leq k$, by recursion, as:

1. $l = 1: (M^0, C) \in \text{Domain}(\Psi_S^{(k)}) \Leftrightarrow (M^0, C) \in \text{Domain}(\Psi_{S,1}^{(1)})$ and $M^1 = \Psi_{S,1}^{(1)}(M^0, C)$.
2. $l = i: M^i = \Psi_{S,i}^{(1)}(M^{i-1}, C)$. ■

PROOF OF LEMMA 4.1

Given the set of modules, M^* , and relations, C^0 , define $\Omega = 2^{M^*} \times 2^{C^0}$ and let Ψ_P^Π be defined as

$$\Psi_P^\Pi(M, C) = \begin{cases} (M, C) & \text{if } \Psi_S(M, C) \neq M_\emptyset \\ (M_\emptyset, C) & \text{otherwise} \end{cases}$$

The set Ω^T is defined as: $\Omega^T = \{(M, C) : \Psi_S(M, C) \neq M_\emptyset\}$. Let the Proper Composition operator be:

$$\Psi_{S_P}(M, C) = \begin{cases} \Psi_S(M, C) & \text{if } (M, C) \in \Omega^T \\ M_\emptyset & \text{otherwise} \end{cases}$$

Now, it is observed - by applying Definition 4.9, and the assumptions of the lemma - that,

$$\Psi_S(M, C) \neq \emptyset \Leftrightarrow \Psi_{S_P}(\Psi_P^\Pi(M, C)) = \Psi_{S_P}(M, C)$$

$$\Psi_S(M, C) = \emptyset \Leftrightarrow \Psi_{S_P}(\Psi_P^\Pi(M, C)) = \Psi_{S_P}(M_\emptyset, C) = M_\emptyset$$

Thus, we conclude that $\forall(M, C) : \Psi_S(M, C) = \Psi_{S_P}(\Psi_P^\Pi(M, C))$. ■

PROOF OF LEMMA 4.2

Proof of Property 1: We prove first for $N = 2$. The correctness for $N > 2$ is verified, by induction on the Order of the Composition operator. Consider a First-Order Composition operator $\Psi_{\bar{S}}^{(1)}$. Let $M^1 \cup M^2 \in \text{Domain}(\Psi_{\bar{S}}^{(1)})$ and infer the following:

$$\Psi_{\bar{S}}^{(1)}(M^1 \cup M^2) = \overline{\bigcup_{M^j \subseteq M^1 \cup M^2, C^j \subseteq C} \langle M^j, C^j \rangle} \supseteq \overline{\bigcup_{M^j \subseteq M^1, C^j \subseteq C} \langle M^j, C^j \rangle} \cup$$

$$\overline{\bigcup_{M^j \subseteq M^2, C^j \subseteq C} \langle M^j, C^j \rangle} \Rightarrow$$

$\Psi_{\bar{S}}^{(1)}(M^1 \cup M^2) \supseteq \Psi_{\bar{S}}^{(1)}(M^1, C) \cup \Psi_{\bar{S}}^{(1)}(M^2, C)$. Consider the property holds for $(k-1)$ -Order Closure Composition operators, let us prove it holds for k -Order Composition operators: for some $\Psi_{\bar{S}}^{(k-1)}$ and $\Psi_{\bar{S}}^{(1)}$; $\Psi_{\bar{S}}^{(k)}(M^1 \cup M^2, C) = \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^1 \cup M^2, C), C)$. Using the induction hypothesis we obtain $\Psi_{\bar{S}}^{(k-1)}(M^1 \cup M^2, C) \supseteq \Psi_{\bar{S}}^{(k-1)}(M^1, C) \cup \Psi_{\bar{S}}^{(k-1)}(M^2, C)$, thus $\Psi_{\bar{S}}^{(k-1)}(M^1 \cup M^2, C) = \Psi_{\bar{S}}^{(k-1)}(M^1, C) \cup \Psi_{\bar{S}}^{(k-1)}(M^2, C) \cup \Delta$, where Δ is properly defined.

Substituting into $\Psi_{\bar{S}}^{(k)}(M^1 \cup M^2, C)$ and applying the induction base, we finally

obtain $\Psi_{\bar{S}}^{(k)}(M^1 \cup M^2, C) = \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^1, C) \cup \Psi_{\bar{S}}^{(k-1)}(M^2, C) \cup \Delta, C) \supseteq$
 $\Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^1, C), C) \cup \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^2, C), C) \cup \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(\Delta, C), C) \supseteq$
 $\Psi_{\bar{S}}^{(k)}(M^1, C) \cup \Psi_{\bar{S}}^{(k)}(M^2, C).$

Proof of Property 2: Let $M^1 \subseteq M^2$. Express M^2 as $M^2 = M^1 \cup (M^2 - M^1)$, and apply Property 1 to obtain: $\Psi_{\bar{S}}^{(k)}(M^2, C) = \Psi_{\bar{S}}^{(k)}(M^1 \cup (M^2 - M^1), C) \supseteq$
 $\Psi_{\bar{S}}^{(k)}(M^1, C) \cup \Psi_{\bar{S}}^{(k)}(M^2 - M^1, C) \supseteq \Psi_{\bar{S}}^{(k)}(M^1, C).$ ■

Proof of Property 3: By induction on the Order of the Composition operator. Consider a First-Order Closure Composition operator $\Psi_{\bar{S}}^{(1)}$. By Definition 4.13, we conclude that,

$$\Psi_{\bar{S}}^{(1)}(M^1, C) = \overline{\bigcup_{M^j \subseteq M^1, C^j \subseteq C} \langle M^j, C^j \rangle} = \overline{\bigcup_{M^j \subseteq M^1, C^j \in (2^C - \{c_1\})} \langle M^j, C^j \rangle} \cup$$

$$\overline{\bigcup_{M^j \subseteq M^1, C^j = \{c_1\}} \langle M^j, C^j \rangle},$$

and by applying Definition 4.5 we conclude $\Psi_{\bar{S}}^{(1)}(M^1, C) \supseteq$

$\Delta \cup M^1 \supseteq M^1$, where the set Δ is properly defined. Consider the property holds for $(k-1)$ -Order Closure Composition operators. Let us prove it holds for k -Order Composition operators. For some $\Psi_{\bar{S}}^{(k-1)}$ and $\Psi_{\bar{S}}^{(1)}$; $\Psi_{\bar{S}}^{(k)}(M^1, C) = \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^1, C), C)$. For $k-1$ we obtain $M^1 \subseteq \Psi_{\bar{S}}^{(k-1)}(M^1, C)$, which implies by Property 2 that,

$\Psi_{\bar{S}}^{(1)}(M^1, C) \subseteq \Psi_{\bar{S}}^{(1)}(\Psi_{\bar{S}}^{(k-1)}(M^1, C), C) = \Psi_{\bar{S}}^{(k)}(M^1, C)$ whilst by the induction base we conclude $M^1 \subseteq \Psi_{\bar{S}}^{(k)}(M^1, C).$ ■

PROOF OF COROLLARY 4.2

For $N = 2$. Since $M^1 \cap M^2 \subseteq M^1$ and $M^1 \cap M^2 \subseteq M^2$, we obtain - by applying the monotonicity property - that,

$\Psi_{\bar{S}}^{(k)}(M^1 \cap M^2, C) \subseteq \Psi_{\bar{S}}^{(k)}(M^1, C)$ and $\Psi_{\bar{S}}^{(k)}(M^1 \cap M^2, C) \subseteq \Psi_{\bar{S}}^{(k)}(M^2, C)$, which finally implies that $\Psi_{\bar{S}}^{(k)}(M^1 \cap M^2, C) \subseteq \Psi_{\bar{S}}^{(k)}(M^1, C) \cap \Psi_{\bar{S}}^{(k)}(M^2, C).$

The induction on N is obvious. ■

PROOF OF LEMMA 4.3

Proof of Property 1: We prove for $N = 2$, the correctness for $N > 2$ is verified, by induction on the Order of the Decomposition operator. Consider a First-Order Decomposition operator $\Psi_{\bar{A}}^{(1)}$. Let $M^1 \cup M^2 \in Domain(\Psi_{\bar{A}}^{(1)})$, and denote

$$\Psi_{\bar{A}}^{(1)}(M^1 \cup M^2) = M^{i(M^1 \cup M^2)}, \Psi_{\bar{A}}^{(1)}(M^1) = M^{i(M^1)} \text{ and } \Psi_{\bar{A}}^{(1)}(M^2) = M^{i(M^2)}.$$

By Remark 4.15.1, $M^{i(M^1 \cup M^2)}$ satisfies $\tilde{\Psi}_{\bar{A}}^{(1)}(M^{i(M^1 \cup M^2)}, C) = (M^1 \cup M^2)$, for some $\tilde{\Psi}_{\bar{A}}^{(1)}$ and C . Definition 4.17 and Remark 4.15.1 imply that there exist sets, \tilde{M}^1

and \tilde{M}^2 , such that: $\tilde{M}^1 \cup \tilde{M}^2 = M^{i(M^1 \cup M^2)}$, $\tilde{\Psi}_{\bar{A}, M^1}^{(1)}(\tilde{M}^1, C) = M^1$ and

$$\tilde{\Psi}_{\bar{A}, M^2}^{(1)}(\tilde{M}^2, C) = M^2. \text{ Now, one can show that } \tilde{M}^1 = M^{i(M^1)} \text{ and } \tilde{M}^2 =$$

$M^{i(M^2)}$, hence $\tilde{M}^1 \cup \tilde{M}^2 = M^{i(M^1 \cup M^2)} = M^{i(M^1)} \cup M^{i(M^2)}$ which implies

$$\Psi_{\bar{A}}^{(1)}(M^1 \cup M^2) = \Psi_{\bar{A}}^{(1)}(M^1) \cup \Psi_{\bar{A}}^{(1)}(M^2). \text{ Consider the property holds for } (k-1)\text{-}$$

Order Closure Decomposition operators, and let us prove it holds for k -Order Decomposition operators. Consider $\Psi_{\bar{A}}^{(k)}$, then by Remark 4.17 we conclude

$$\Psi_{\bar{A}}^{(1)}(M^1 \cup M^2) = \Psi_{\bar{A}}^{(1)}(\Psi_{\bar{A}}^{(k-1)}(M^1 \cup M^2)). \text{ For } k-1, \text{ we obtain by applying the}$$

induction hypothesis that $\Psi_{\bar{A}}^{(k-1)}(M^1 \cup M^2) = \Psi_{\bar{A}}^{(k-1)}(M^1) \cup \Psi_{\bar{A}}^{(k-1)}(M^2)$. Thus,

substituting into $\Psi_{\bar{A}}^{(k)}(M^1 \cup M^2)$, and applying the induction base, we conclude

$$\text{that } \Psi_{\bar{A}}^{(k)}(M^1 \cup M^2) = \Psi_{\bar{A}}^{(1)}(\Psi_{\bar{A}}^{(k-1)}(M^1) \cup \Psi_{\bar{A}}^{(k-1)}(M^2)) = \Psi_{\bar{A}}^{(1)}(\Psi_{\bar{A}}^{(k-1)}(M^1))$$

$$\cup \Psi_{\bar{A}}^{(1)}(\Psi_{\bar{A}}^{(k-1)}(M^2)) = \Psi_{\bar{A}}^{(k)}(M^1) \cup \Psi_{\bar{A}}^{(k)}(M^2). \quad \blacksquare$$

The proofs of Properties 2-4 are immediately obtained.

PROOF OF COROLLARY 4.4

Apply similar argument as in Corollary 4.2.

PROOF OF OBSERVATION 4.3

Let $M^{(k-1)} = \Psi_A^{(k-1)}(M)$, and consider $\Psi_A^{(k)} = \Psi_{A,k}^{(1)} \circ \Psi_A^{(k-1)}$. If $M^{(k)} = \Psi_A^{(k)}(M)$, it is easy to show that the strict regularity of M^* (Definition 4.7) implies that $Order(M^{(k)}) < Order(M^{(k-1)})$, thus there exists a finite integer $k^*(M)$, such that $Order(\Psi_A^{(k^*(M))}(M)) = 0$, which means that: $(\Psi_A^{(k^*(M))}(M)) = \{ \bar{M} : \bar{M} \subseteq M_0 \}$. ■

PROOF OF LEMMA 4.4

Let us assume that $m \in M_k$. Define a first-order decomposition operator $\Psi_A^{(1)}$, such that $\Psi_A^{(1)}(m)$ is the carrier set of m . Let us extend $\Psi_A^{(1)}$ to a set M by letting $\Psi_A^{(1)}(M)$ be the union of the carrier sets of each module in M . It is easily seen that $\exists k^* : \Psi_A^{(k^*)} = \Psi_A^{(1)} \circ \Psi_A^{(1)} \circ \dots \circ \Psi_A^{(1)}$ satisfies $\Psi_A^{(k^*)}(m) = \tilde{M} \subseteq M_0$; M_0 is the class of basic modules. Following Remark 4.15.1, $\Psi_A^{(k^*)}$ has a 'companion' composition operator denoted by Ψ_S , such that $m = \Psi_S(\tilde{M}, C)$. Finally by Observation 4.1, $\Psi_S^{(k)}$ may be decomposed in form $\Psi_S^{(k)} = \Psi_{S,k}^{(1)} \circ \Psi_{S,k-1}^{(1)} \circ \dots \circ \Psi_{S,1}^{(1)}$; which concludes the proof. ■

PROOF OF OBSERVATION 4.4

Proof of Property 1: By Definitions 4.19 and 4.20, $\Psi_{\bar{C}}^{(k_1, k_2)} = \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)}$ for some Composition and Decomposition operators. Let us show that $\Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)} = \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)}$. It is easily inferred, by applying Lemma 4.2 (2) and Lemma 4.3 (2), that $\Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C) \supseteq \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C) \supseteq \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C)$. But Definition 4.20 implies that:

$$\Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)} \subseteq \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)}; \text{ thus } \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)} = \Psi_{\bar{S}}^{(k_1)} \circ \Psi_{\bar{A}}^{(k_2)}. \quad \blacksquare$$

Proof of Property 2: By Property 1, $\Psi_{\bar{C}}^{(k_1, k_2)} = \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C)$. Corollary 4.3 implies that $\Psi_{\bar{A}}^{(k_2)}(M) \subseteq \Psi_{\bar{A}}^{(\tilde{k}_2)}(M)$. Thus, applying Lemma 4.2 (2), we conclude

that $\Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C) \subseteq \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(\tilde{k}_2)}(M), C)$, which concludes the proof. ■

Proof of Property 3: By Property 1, $\Psi_{\bar{C}}^{(k_1, k_2)} = \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C)$. Thus applying Corollary 4.1, we conclude that $\Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C) \subseteq \Psi_{\bar{S}}^{(\tilde{k}_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C)$. ■

Proof of Property 4: By applying Definition 4.21 and Property 1, one obtains

$$\Psi_{\bar{C}}(M, C) = \bigcup_{k_1, k_2} \Psi_{\bar{C}}^{(k_1, k_2)} = \bigcup_{k_1} \bigcup_{k_2} \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}^{(k_2)}(M), C) \subseteq$$

$$\bigcup_{k_1} \Psi_{\bar{S}}^{(k_1)}(\bigcup_{k_2} \Psi_{\bar{A}}^{(k_2)}(M), C),$$

where the latter inclusion is inferred by applying

Lemma 4.2 (1). Now, by applying Definition 4.13 and Definition 4.18, we obtain

$$\bigcup_{k_1} \Psi_{\bar{S}}^{(k_1)}(\bigcup_{k_2} \Psi_{\bar{A}}^{(k_2)}(M), C) = \bigcup_{k_1} \Psi_{\bar{S}}^{(k_1)}(\Psi_{\bar{A}}(M), C) = \Psi_{\bar{S}}(\Psi_{\bar{A}}(M), C). \quad \blacksquare$$

APPENDIX B (BASIC NOTIONS OF SET THEORY)

This appendix defines the set theory terms used in this chapter. The terminology used in this book is at the introductory level of set theory.

We use the usual notation of set theory. Sets are often defined by a property and we write $\{x: x \text{ has property } P\}$ to denote the set consisting of all those elements which enjoy the property P . The terms “collections,” “aggregate,” and “family” are synonyms for set; the members of a set are often referred to as its “elements” or “objects.” These terms are used in this book in a way which agrees with their customary usage.

Sets are usually denoted by capital letters and elements by lower case letters. The symbol “ \in ” (abbreviation for “belongs to”) indicates set membership. Thus $a \in A$ means that object a is a member of set A , and $b \notin A$ means that b is not a member of A .

The set having no members is called the *empty set* and is denoted by the symbol \emptyset .

If A and B are sets for which each member of A is also a member of B , then A is a subset of B or is said to be *contained in* B . Such set inclusion is denoted by the symbol “ \subseteq ”: $A \subseteq B$ provided that A is a subset of B . The concept of set inclusion defined here allows for the possibility of equality: $A \subseteq B$ includes the possibility $A = B$. $A \subseteq B$ is read A is *contained in or equal to* B . In some texts, this relation is expressed $A \subset B$.

Observe that $A \subseteq A$ and $\emptyset \subseteq A$ for every set A . The latter inclusion is true because \emptyset has no members and therefore has no members outside A . Two sets A and

B are equal precisely when each is a subset of the other. This fact is often used in showing set equality.

It is often desirable to discuss collections of sets and to name many sets in a systematic way. The standard method of doing this, called *indexing*, is defined next. Let A be a set for which, corresponding to each element $a \in A$, there is a set M_a . Then the collection of sets $\{M_a : a \in A\}$, also denoted $\{M_a\}_{a \in A}$, is said to be *indexed* by A or to have A as *index set*.

A note on word usage is in order before defining the standard set operations. The conjunction "or" is used in mathematics and logic in the inclusive sense: If p and q are statements, then the statement " p or q " is true whenever at least one of p , q , is true. The only case for which " p or q " is false is the case in which p is false and q is also false.

If A and B are sets, the *union* $A \cup B$ of A and B is the set consisting of all elements x which belong to at least one of the sets A, B :

$$A \cup B = \{x: x \in A \text{ or } x \in B\}.$$

The *intersection* $A \cap B$ of A and B is the set of all elements x which belong to both A and B , that is, the set of elements common to A and B :

$$A \cap B = \{x: x \in A \text{ and } x \in B\}.$$

Sets A and B are said to be *disjoint* if $A \cap B = \emptyset$.

The *Cartesian product* of two sets A, B is denoted $A \times B$: $A \times B = \langle a, b \rangle$, and $a \in A$ and $b \in B$. We often use A^n to denote $A \times A \times \cdots \times A$ (n times), and an element of A^n is written $\langle a_1, \dots, a_n \rangle$. The definition of Cartesian product is easily extended to more than two factors. If $\{A_i\}_{i=1}^n$ is a finite sequence of sets, then their Cartesian product, denoted by $A_1 \times A_2 \times \cdots \times A_n$ or by, $\prod_{i=1}^n A_i$, is defined by

$$\prod_{i=1}^n A_i = \{(a_1, a_2, \dots, a_n) : a_i \in A_i \text{ for each } i = 1, 2, \dots, n\}.$$

A *function* f from A to B , written $f : A \rightarrow B$, is a mapping of the elements of A to the elements of B , i.e., it associates with each $a \in A$ a unique element $f(a) \in B$. It can be viewed as a subset of $A \times B$ which satisfied:

1. for every $a \in A$ there is $b \in B$ such that $\langle a, b \rangle \in f$,
2. $\langle a, b \rangle \in f$ and $\langle a, b' \rangle \in f$ implies $b = b'$.

For a subset A' of A , the set $f(A') = \{b \in B : b = f(a) \text{ for some } a \in A'\}$ is called the *image* of A' under f . The set $f(A)$, the image of the domain under f , is sometimes called the *image* of the function.

For a subset B' of B , the set $f^{-1}(B') = \{a \in A : f(a) \in B'\}$ is the *inverse image* of B' under f . The set of points $\{\langle a, b \rangle \in A \times B : b = f(a)\}$ is called the *graph* of the function f .

A function f is *surjective* if for every $b \in B$ there exists some $a \in A$ such that $f(a) = b$; it is *injective* if $f(a) = f(a')$ implies $a = a'$. $f : A \rightarrow B$ is a *one-to-one correspondence* (or a *bijection*) if it is both surjective and injective. In this case there is an *inverse function* $f^{-1} : B \rightarrow A$ which assigns to each b in B its unique pre-image $a = f^{-1}(b)$ in A .

We use the usual notion of *composition* of functions. If $f : A \rightarrow B$ and $g : B \rightarrow C$, then $g \circ f \rightarrow C$ is the function defined by $g \circ f(a) = g(f(a))$ for every $a \in A$.

REFERENCES

1. Appel, A., "Modeling in Three Dimensions," *IBM Systems Journal*, Vol. 7, 3-4 (1968).
2. Braid, I.C., "The Synthesis of Solids Bounded by Many Faces," *Communications of the ACM*, Vol. 18, 4 (1975).
3. Lavin, M.A., and Lieberman, L.I., "A System for Modeling Three Dimensional Objects," *IBM Research Report RC-5765*, (1975).
4. Ulrich, K.T., "Computation and Pre-Parametric Design," *Technical Report 1043*, MIT Artificial Intelligence Laboratory, 1988.
5. Vemuri, R.K., Soo-Ik Oh, and Miller, R.A., "Topology-Based Geometry Representation to Support Geometric Reasoning," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 19 (2) (1988).

CHAPTER 5

THE IDEALIZED DESIGN PROCESS

This chapter attempts to scientifically define important aspects of the design process including: the feasibility to design, the role of the designer in the design process, the human-machine interface, and the abilities of a human designer. The intuitive concept of the design process as a mapping from the desired function and constraints, called specifications, to the artifact description is formalized in this chapter by introducing the notion of an *idealized design process*. The function and attribute spaces are represented by propositional calculus (see Appendix A). The principle of design consistency; which, roughly speaking, states that *small changes in specifications should lead to small changes in design* (and vice versa) is formalized by introducing the notion of a continuous mapping of one closure space to another. The concept of a *basis* for the artifact and function spaces is introduced, and its relation with the principle of design consistency is explored.

5.1 INTRODUCTION

In Chapter 4, a general model for the representation of design artifacts (constituting the attribute space) was constructed. The primary concept of FDT used in this chapter is that the design process is a mapping (see Figure 5.1) of the desired set of *specifications* (describing the desired functions and constraints of the final product) onto the *artifact description* (the final detailed product description). This synthesis process constitutes a mapping between the *functional* and *physical* domains. In turn, the attributes generated in the physical domain are interpreted as a set of specifications for implementation (in terms of process variables) in the *process* domain. Some instances of this mapping include: (1) an information retrieval system by which the user of the system can search for the appropriate artifact that satisfies the specified requirements; (2) a mapping from the desired function and constraints to the artifact description; both embedded in an Euclidean space that can be described by mathematical equations; (3) a rule based system that includes a set of rules; each of the form IF function THEN structure, which produce a solution in terms of attributes by giving a specification in terms of function; and (4) an evolutionary design process (as described in Chapter 6) that progresses from a set of specifications and moves towards the *artifact description* (the final detailed product

description).

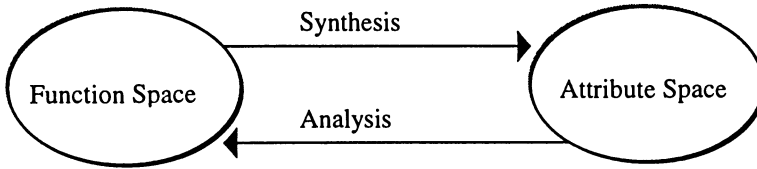


Figure 5.1 Design as a Mapping between Function and Attribute Spaces

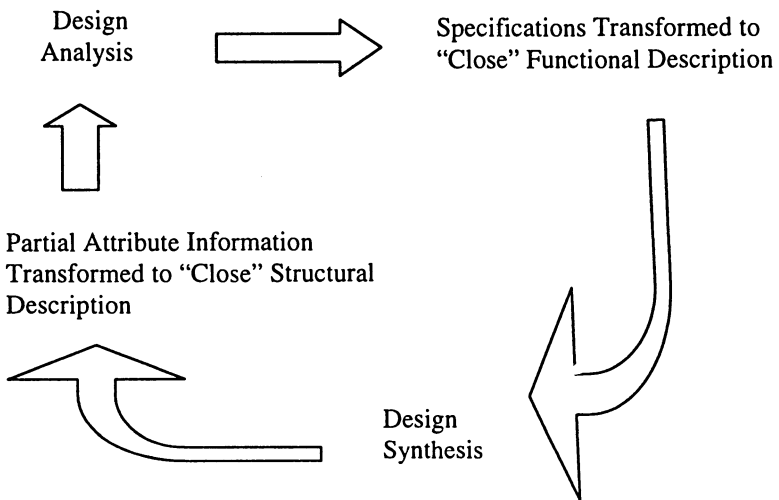


Figure 5.2 The Idealized Design Process Cycle

An idealized design process includes several steps (see Figure 5.2):

1. The different attributes included in the attribute space are mapped to functional properties that form the function space. This step, which provides a basis for reasoning at the synthesis stage, conceptualizes the issue of design *analysis*;
2. The desired set of specifications (describing the desired functions and constraints of the final product) is transformed to a “close” functional description that is sufficiently detailed. A “close” functional description represents the concept of a *model* as mediating between the function and attribute spaces. Models that are sufficiently detailed can be used to support the incremental modification of the design towards a solution. For example, if we design mechanical machines, the functional specification will be transformed into a graphic representation of the machine to be designed, and then the model will be transformed into the

structural description of the machine.

3. The detailed functional description of the artifact is mapped to the attributes or the structure of the artifact (part of the attribute space) by the “inverse” of the analysis mapping (i.e., *synthesis*). These attributes are observable properties that are needed in order to manufacture the artifact;
4. If we are able to locate only partial attribute information for the candidate solution, then the partial information is transformed into a “close” structural description. The “close” structural description is sufficiently detailed to provide the necessary manufacturing information for the artifact.

The state of the idealized design process is characterized by properties that convey the assumptions of the theory. These assumptions reflect the nature of objects and their potential manipulation to achieve a desired functionality, the role of the designer in the design process, the human-machine interface, the abilities of a human designer, and the feasibility to design. These properties are the foundations of the theorems discussed later.

To formulate and analyze important aspects of the design process, we emphasize the application of topological ideas to design. The main purpose of this chapter is to investigate the notion of design consistency: *small changes in specifications should lead to small changes in design* and vice versa. The mathematical concept that is used to investigate the principle of design consistency is continuity (*continuous analysis* and *continuous synthesis*). Continuity is a process-oriented concept. It guarantees that a small change in the artifact description will result in a small change in the artifact functionality and vice versa. Therefore, if the current candidate’s functionality differs slightly from the required function, a small modification to the structure may be sufficient in order to satisfy the altered function.

There are several important properties of continuous mapping of interest in design tasks such as synthesis and analysis: *metric*, *convergence*, *homeomorphism* and *basis*.

If a space is *metric*, then one can calculate the *distance* between any two entities in the function and attribute spaces. The distance measure can be used to assign *values* to each of the attributes or functions describing an artifact.

Convergence is also a process-oriented concept; it provides a different perspective on continuity. Convergence guarantees that a sequence of incremental refinement changes to artifact functionality will cause only small incremental changes to structure.

A transformation between the function space and attribute space that conserves the continuity or convergence properties is useful in design; because, it allows the creation of different viewpoints of the desired functionality and the partial design description that may simplify or direct future steps. In this case, both the analysis and synthesis (the “inverse” of analysis) mappings are continuous. In topology, such a transformation is called *homeomorphism*. One objective of this chapter is to provide a list of design properties that are not destroyed by continuous transformations between function and attribute spaces. Even though the concept of continuity and homeomorphism is useful in design, we show that they are often not easy to check.

The intuitive meaning of a *basis* for a function space (or attribute space) is the collection of already decomposable functions (or attributes) that can be utilized to decompose a new function. Thus, the designer can utilize the basis functions (attributes), which have already been solved, to more easily solve a new problem.

Several design theories exist that more formally describe design as the designation of a domain in the attribute space, which corresponds to a domain that designates the specification. Yoshikawa's general design theory (GDT) mathematically describes the design process in terms of point-set topology or set-theoretic topology, building up from design axioms [1, 2, 3, 4]. GDT attempts to scientifically compute important aspects of the design process including: the role of a designer in the design process, the man-machine interface, the abilities of a human designer, and the possibility to design. The theory does not, however, address how this process happens in real design. GDT restricts the nature of knowledge to two perfect properties: (1) GDT assumes that the function and attribute spaces are topologies of the set of all real objects that existed in the past; exist in the present, or will exist in the future (the entity set); and (2) the state of design knowledge is characterized by the ability to separate between any two entities. Through these properties, GDT guarantees the termination of the design process.

Yoshikawa's general design theory does not hold for real design processes for the following reasons. First, the refinement process is made easier by the use of the entity set as mediator between the specification and the design description. In the absence of the entity set the process could be more complex. GDT applies to domains with topological structure, but real domains do not satisfy this requirement. Moreover, the restriction to domains with topological structure limits the design selection to the entity set (catalogue). The second reason that GDT does not apply to real design is because all entities have the same status under the assumption of a topological structure of the entity set. However, it is recognized that in real design, the overall organization of concepts and entities is hierarchical.

The "ideal design process" presented in this chapter broadens the scope of Yoshikawa's theory by insisting on less restrictive assumptions: (1) the design process is a mapping of the desired functionality of a product onto the description of the final product *without the intervention of the entity set*; and (2) human designers use hierarchical knowledge structure for the overall organization of functional and structural properties. To this end, the idealized design process attempts to cast these assumptions in the framework of "closure" topological spaces (as opposed to point set topology), and uses this framework to prove theorems about the nature of design. As such, a closure topological structure of functions and attributes provides an interesting perspective for viewing design.

This chapter is organized as follows: Section 5.2 describes the domain of mechanical fasteners and car horns, which are used to illustrate the ideas discussed in the chapter. Section 5.3 reviews the concepts of the idealized design process. Section 5.4 conveys the idealized design process assumptions (axioms) about the nature of the mapping from the desired function and constraints to the artifact description. These assumptions are the foundations of the theorems discussed in the chapter. Section 5.5 introduces the notion of a basis for attribute and function spaces, and

suggests that the “smallest” basis can gauge the descriptive complexity inherent in a space. Section 5.6 summarizes the chapter. For reading fluency, we present the proofs of the theorems in Appendix D.

5.2 MOTIVATING SCENARIOS

In this section we describe two domains that are used to illustrate and explain the concepts discussed in this chapter.

5.2.1 MECHANICAL FASTENERS

Figure 5.3 depicts seven mechanical fasteners (see also Chapter 3.2). They will be referred to as the *fasteners domain*. Each fastener in the figure is denoted by a letter. The fasteners have properties that can be observed, and are termed *structural attributes* of the artifact. Some of these attributes are summarized in Table 5.1. In the table, a “1” denotes that a fastener has the corresponding structure; a “-” denotes that it does not. Additional observable properties that fasteners may have but that are not mentioned include *nominal size* (fraction diameter), *number of threads per inch*, *thread form* (the configuration of the thread in an axial plane), and *thread fit* (that specifies the allowance between the nut and the bolt).

In addition to providing structures, each fastener provides some *functionality* as summarized in Table 5.2. Additional functions that fasteners may have but that are not mentioned included *locking ability*, *prevailing torque*, *sealing*, *thermal expansion*, *electrical resistance*, *thermal conductivity*, and *magnetic susceptibility*.

Table 5.1 Structural Properties of Fasteners

<u>structure</u>	<u>fastener</u>						
	A	B	C	D	E	F	G
slot drive recess	0	0	0	0	0	1	0
cap drive recess	0	1	0	0	0	0	0
external socket hex	0	0	0	1	0	0	0
cap head type	0	1	0	0	0	0	0
round head type	0	0	0	0	0	1	0
truss head type	0	0	0	0	1	0	0
shoulder	0	1	0	1	1	1	0
threaded shank	0	1	0	1	1	1	0
threaded tail	0	1	0	1	1	1	0
chamfer point	0	1	1	1	1	1	1
fastener with washer	0	0	0	0	0	0	1
polycarbonate material	1	1	1	1	0	0	1

Table 5.2 Functional Properties of Fasteners

function	fastener						
	A	B	C	D	E	F	G
high strength	0	1	0	1	1	1	0
retractability	0	1	0	1	1	1	0
actuation ease	1	0	0	0	1	0	0
head out	0	1	0	1	1	1	0
tail out	1	0	1	0	0	0	1
medium precision	0	1	0	1	1	1	0
corrosion resistance & thermal conductivity	1	1	1	1	0	0	1

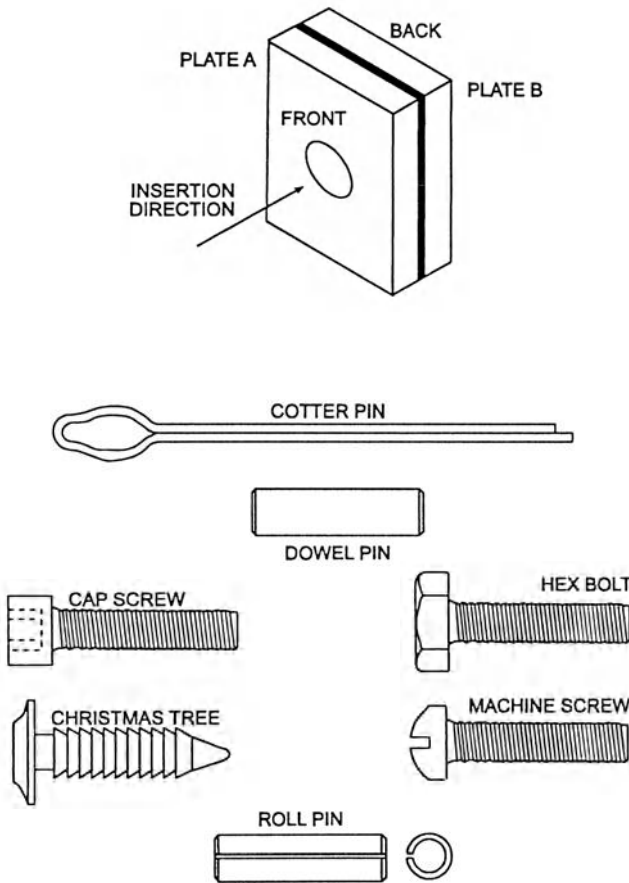


Figure 5.3 Seven Fasteners in the Knowledge Base (adapted from [11])

There are functions that are directly derived from the structure of a fastener. For example, *polycarbonate* fasteners (such as *copper-alloy* fasteners) are used where problems of *corrosion* and *thermal conductivity* exist, or a fastener that has *threaded tail* has *medium precision* in laterally locating plate A with respect to plate B. Note that this structure-function relation may be an approximation. Other functions may be more complex and can not be inferred from one observable property. For example, fastener E (Christmas tree fastener) provides *actuation ease* although it does not have a drive rotary mode. Some functions may qualify other functions; for example, the function *retractability* qualifies the function *head out*.

The ability to infer functionality from artifact structure is useful in *analysis*. In contrast, generating artifact structure to satisfy a desired function is related to *synthesis* (which is the primary focus of this book). For example, the fastener specifications *high strength* and *actuation ease* lead to one potential design: fastener E. This design solution can be generated in the following way. We start with {B, D, E, F} as the *high strength* designs and refine them with the *actuation ease* property. The refinement process is made easier by the use of the seven case fasteners as intermediaries between the specifications and the design description. In the absence of these case fasteners, the synthesis process could be more difficult.

As another example, assume that in addition to the previous specifications, it is also required that the fastener have the property *corrosion resistance & thermal conductivity*. There is no fastener that satisfies the three functions. A redesign process could be invoked by taking the current candidate design E and retracting either the *high strength* or the *actuation ease* specifications and then trying the new specification. Alternatively, if the set of designs is not confined to the seven fasteners, fastener E has the property *corrosion resistance & thermal conductivity* by using *copper-alloy* material.

We may not be able to find any candidate that satisfies all three specification properties. Nevertheless, we have two sets of nearly good candidates: (1) *high strength* and *actuation ease* {E}; (2) *high strength* and *corrosion resistance & thermal conductivity* {B, D}; and (3) *actuation ease* and *corrosion resistance & thermal conductivity* {A}.

5.2.2 THE CAR HORN

A typical energy-storing transducer (transducers are devices that couple distinct energy domains) of practical interest is the condenser electrostatic horn. The sketch of Figure 5.4 shows roughly how these devices may be constructed. A capacitor is formed by mounting a movable plate near a rigidly mounted plate and providing electrical connections as one would in a conventional parallel-plate capacitor. In practice, the movable plate might be a thin diaphragm or tightly stretched membrane on which a thin layer of conducting material is fixed. Such distributed parameter “plates” can move in complicated ways.

Figure 5.5 depicts seven comparable concepts for car horn. A car horn is an audible means of warning of the presence of the car. The car horns have *structural*

attributes that can be observed; some of these are summarized in Figure 5.5. In addition to providing structures, each car horn provides some *functionality* that is summarized in Table 5.3.

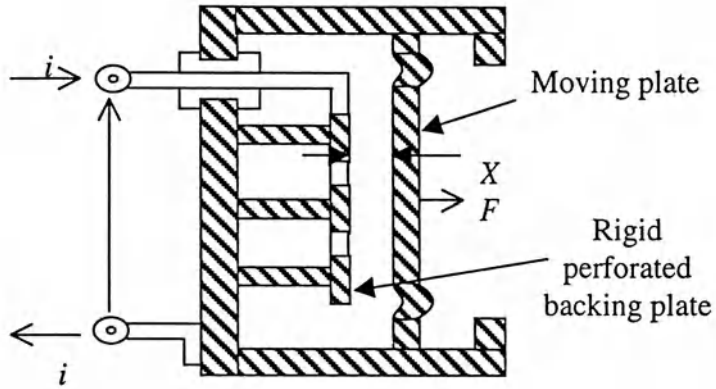


Figure 5.4 A Typical Energy-storing Transducer: The Condenser Microphone or Electrostatic Loudspeaker

Table 5.3 Functional Properties of Car Horns

function	car horns						
	A	B	C	D	E	F	G
ease of achieving 105-125 DbA	1	1	1	0	0	0	1
ease of achieving 2000-5000 Hz	1	1	1	1	0	0	1
resistance to corrosion, erosion and water	1	0	1	0	0	0	1
resistance to vibration, shock and acceleration	1	1	0	0	0	0	0
resistance to temperature	1	0	0	1	0	0	1
high response time	1	0	0	0	0	0	0
small number of stages	1	1	0	0	1	1	0
low power consumption	1	0	1	0	0	0	1
ease of maintenance	1	1	0	0	1	1	0
low weight	1	0	0	1	0	0	1
small size	1	0	0	0	0	0	0
small number of parts	1	1	0	0	0	0	0
long life in service	1	1	0	0	0	0	0
low manufacturing cost	0	1	0	0	0	0	0
ease of installation	1	1	0	1	0	0	0
long shelf life	1	1	0	1	1	1	1

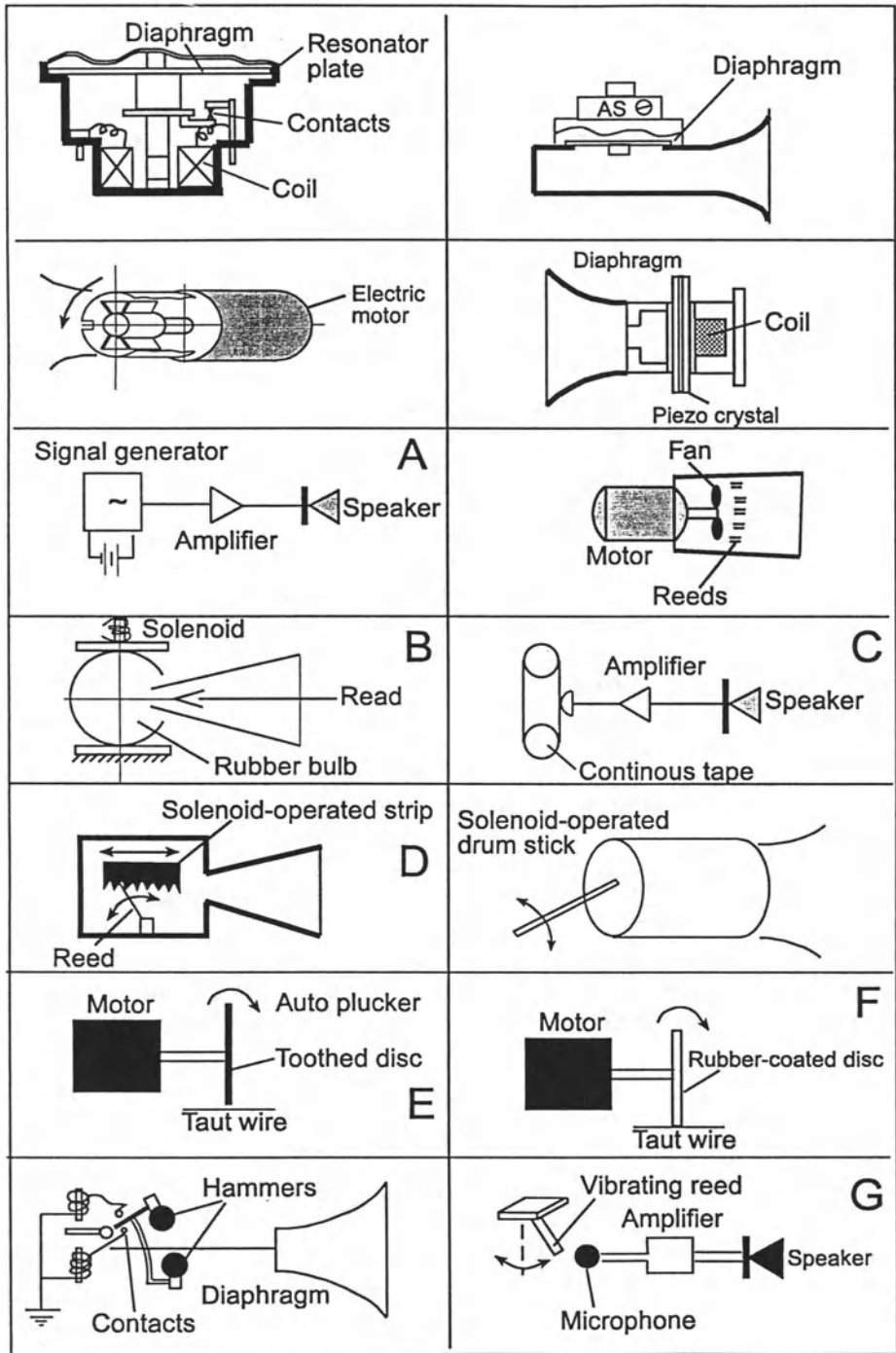


Figure 5.5 Comparable Concepts for Car Horn (adapted from [12])

5.3 PRELIMINARIES

This section reviews the idealized design process terminology and definitions; including simple working examples (such as the mechanical fasteners and car horn domain). The theory attempts to cast design in the framework of Propositional calculus and closure topological spaces (see Appendix A). We start with assumptions about the nature of structural and functional attributes, and use them in the following sections to prove theorems about design.

5.3.1 THE ATTRIBUTE AND FUNCTION SPACES

Definition 5.1: An attribute is any structural property (e.g., physical, mechanical, geometrical, or chemical) that can be observed or measured by scientific means (e.g., through the use of an instrument). An artifact has respective values for its attributes. Each of the attributes can be perceived as *manufacturing information* (such as dimensions, tolerances, and manufacturing techniques).

Example 5.1: The fasteners have properties that can be observed and therefore describe the structural attributes (e.g., *slot drive recess*, *round head type*, or *chamfer point*) of the artifact; some of these are summarized in Table 5.1. A value of “1” in the table denotes that a fastener has the corresponding structure; a “-” denotes that it does not.

Definition 5.2: A functional property is the behavior that an artifact displays when it is subjected to a situation. The collection of all functions observed in different situations is the functional description of the artifact.

Example 5.2: The fastener properties listed in Table 5.2 are all functional properties, and thus can be observed. For example, *corrosion resistance & thermal conductivity* can be manifested as a behavior when a fastener is immersed in water. For instance, fastener E corrodes in water, and thus, does not have the functional property *corrosion resistance & thermal conductivity*. For fastener E, the lack of this functional property is a direct consequence of the non-polycarbonate material that is used.

Definition 5.3 (Basic Attributes, M_0): Basic attributes (properties) cannot be defined in terms of other attributes. The set M_0 represents basic properties that can be assembled to construct the set of all artifact descriptions within the domain.

Example 5.3: The fastener attributes summarized in Table 5.1; e.g., *slot drive recess*, *cap drive recess*, *external socket hex*, *cap head type*, *round head type*, *threaded tail*, etc.

Definition 5.4 (Basic Functions, F_0): Basic functions cannot be defined in terms

of other functions. The set F_0 represents basic functional properties that can be assembled to construct the set of all functional descriptions within the domain.

Definition 5.5 (Attribute Space, \mathcal{D}): The attribute space is the tuple $\mathcal{D} = \langle M_0, C^0, M^* \rangle$; where $C^0 = \{\wedge, \vee, \sim\}$, and M^* is the set of *all* artifact descriptions (or attributes) obtained by conjunction \wedge (logical AND), disjunction \vee (logical OR), and negation \sim over the basic attributes (i.e., M_0) in \mathcal{D} . By convention, we use $m \in \mathcal{D}$ instead of $m \in M^*$.

Example 5.4: Let us denote the basic fastener attributes summarized in Table 5.1, respectively, by m_1, m_2, \dots, m_{12} . For example, m_3 designates the basic attribute *external socket hex drive recess*. In particular, the composite attribute $m_1 \wedge \sim m_2 \wedge \sim m_3 \wedge \sim m_4 \wedge m_5 \wedge \sim m_6 \wedge m_7 \wedge m_8 \wedge m_9 \wedge m_{10} \wedge \sim m_{11} \wedge \sim m_{12}$ designates a description of fastener F as specified in Table 5.1, and is included in the attribute space \mathcal{D} .

Definition 5.6 (Function Space, \mathcal{F}): The function space is the tuple $\mathcal{F} = \langle F_0, C^0, F^* \rangle$; where $C^0 = \{\wedge, \vee, \sim\}$, and F^* is the set of *all* functional properties (functional descriptions) obtained by conjunction (\wedge), disjunction (\vee), and negation (\sim) over the basic functional properties (i.e., F_0) in \mathcal{F} . By convention, we use $f \in \mathcal{F}$ instead of $f \in F^*$. The *design specification* designates the function of the required artifact by using functional properties.

Example 5.5: Let us denote the basic fastener functional properties summarized in Table 5.2, respectively, by f_1, f_2, \dots, f_7 . For example, f_3 designates the basic functional property *actuation ease*. In particular, the composite functional property $\sim f_1 \wedge \sim f_2 \wedge f_3 \wedge \sim f_4 \wedge f_5 \wedge \sim f_6 \wedge f_7$ designates the functional description fastener B as specified in Table 5.2, and is included in the function space \mathcal{F} .

Example 5.6: In parametric design, an artifact is described in terms of a finite set of attributes (a feature of a part-type that other part-types can connect to or receive information through; e.g., axial pitch). Each attribute can be described by its dimension. Since it is difficult and very costly to manufacture a device or build a structure with exact desired dimensions, designers use *tolerances* to specify the permissible variation in size and shape. Tolerances are an integral part of the design documents. In fact, tolerances are intimately linked to each specified dimension in the detailed drawings. By specifying a tolerance, the designer makes it possible for the manufacturing (or building) inspector to verify that the particular components of the designed product fall within the allowed limits. The *tolerance limits* of a dimension are the largest and the smallest that a part can be. For example, when a designer specifies that the finished size of a part has to fall between 3.8883" and 3.875," the

quality inspector rejects parts that do not fall within those limits. Functional properties are also specified in terms of *tolerances* (e.g., “produce for less than a given cost”) that specify the permissible variation in the artifact’s functionality. Functional properties are represented by a fully-constrained system of equal constraints as defined by the user. The constraints are defined in terms of the tolerances linked to each specified dimension of the part to be created.

At the beginning of the design, the designer is provided with a design specification that is described in terms of tolerances that specify the permissible variation in the artifact’s functionality. The designer is also given a set of constraints that, when solved, produces the desired tolerances; each tolerance intimately linked to a specified dimension.

This type of interaction between the function and attribute spaces can be described by solving systems of interval equations as described by interval analysis techniques (see Chapter 14). To establish necessary terminology, an attribute or a functional property is described in terms of closed intervals: $[a, b] = \{x \in \mathfrak{R} : a \leq x \leq b\}$. $[a, b]$ is true if the respective dimension x of the attribute (or functional property) satisfies $x \in [a, b]$. The system of intervals can be considered a logical conjunction, disjunction, and negation as follows: $[a, b] \wedge [c, d]$ is true if the respective dimension $x \in [a, b] \cap [c, d]$; $[a, b] \vee [c, d]$ is true if the respective dimension $x \in [a, b] \cup [c, d]$; and $\sim[a, b]$ is true if the respective dimension $x \notin [a, b]$.

Designing in established design domains may vary from the simple selection of artifacts from a catalogue to the routine formation of composing artifacts from available components. The next definition corresponds to the situation of choosing a design solution from a catalogue.

Definition 5.7 (Artifact Catalogue, \mathcal{S}): The set of all real artifacts that existed, and exist in a catalogue. An artifact in \mathcal{S} is a *design solution*, which means its functional and structural description are included in the function space and attribute space, respectively. Therefore, a design solution satisfies some requirements and contains the necessary manufacturing information. If there is a *one-to-one* correspondence between an artifact in the catalogue \mathcal{S} and its functional (or structural) representation, we shall refer to the functional description of the artifact as the artifact.

Example 5.7: The specifications of a fastener that will have *high strength* and *actuation ease* leads to one potential design solution in \mathcal{S} : fastener E. The attributes that correspond to this design solution that appear in Table 5.1 are: *truss head type, shoulder, threaded shank, threaded tail, and chamfer point*.

5.3.2 PROXIMITY IN FUNCTION AND ATTRIBUTE SPACES

The next definition formalizes the intuitive notion of proximity in function space.

Definition 5.8 (Closure Function Space): If $U_{\mathcal{F}}$ is a single-valued relation on $2^{\mathcal{F}}$ with a range of $2^{\mathcal{F}}$, then we say that $U_{\mathcal{F}}$ is a *closure operation* (or “closure”) for \mathcal{F} provided that the following conditions are satisfied:

1. $U_{\mathcal{F}}(\emptyset) = \emptyset$
2. $F \subset U_{\mathcal{F}}(F)$ for each $F \subset \mathcal{F}$;
3. $U_{\mathcal{F}}(F \cup G) = U_{\mathcal{F}}(F) \cup U_{\mathcal{F}}(G)$ for each $F \subset \mathcal{F}$ and $G \subset \mathcal{F}$;

If f is a functional property (or design specification) in \mathcal{F} , then $U_{\mathcal{F}}(f)$ designates a set of functional descriptions in \mathcal{F} such that each functional description derives (by means of *logical inference*) the functional property f . Each element in the set represents a different “viewpoint” of the design functionality f . If F is a set of functional properties in \mathcal{F} , then $U_{\mathcal{F}}(F) = \bigcup_{f \in F} U_{\mathcal{F}}(f)$ (in which case $U_{\mathcal{F}}$ is called *additive*).

The structure $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is called a *closure function space*. In the closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, the functional property x is *proximal* to F if and only if $x \in U_{\mathcal{F}}(F)$.

Definition 5.9 (Topological Function Space): A *topological closure operation* (or “topological closure”) for a set \mathcal{F} is a closure operation $U_{\mathcal{F}}$ for \mathcal{F} satisfying the following condition: $F \subseteq \mathcal{F}$ implies $U_{\mathcal{F}}(U_{\mathcal{F}}(F)) = U_{\mathcal{F}}(F)$. $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is called a *topological function space*.

If $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is a topological function space, then for every functional property f , $U_{\mathcal{F}}(f)$ designates the set of *all* functional descriptions in \mathcal{F} such that each functional description derives (by means of *logical inference*) the functional property f .

Example 5.8: Let \mathcal{F} be the set of all functional descriptions of car horns obtained by conjunction (\wedge), disjunction (\vee), and negation (\neg) over the basic functional properties in Table 5.3. If the set F contains the single functional property *low power consumption*, then the *closure* of F in $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ or under $U_{\mathcal{F}}$ contains, among others, the following proximal functional properties (each functional property derives *low power consumption*): (1) *low power consumption*; (2) *low power consumption* \wedge *ease of maintenance*; (3) *low power consumption* \wedge \neg *ease of maintenance*; and (4) *low power consumption* \wedge (*resistance to corrosion & erosion and water* \vee *resistance to vibration*) \wedge (*shock and acceleration* \vee *resistance to temperature*). Each proximal functional property represents an alternative “viewpoint” of *low power consumption*.

Example 5.9: The designer may realize that *high strength* of fasteners is

characterized separately by the head and tail of the fastener. By applying the knowledge-base of fasteners and analyzing the current specification, it is concluded that both of them require high strength; i.e., $high\ head\ strength \wedge high\ tail\ strength \Rightarrow high\ strength$. Thus, $\{high\ head\ strength \wedge high\ tail\ strength\} \in U_{\mathcal{F}}(\{high\ strength\})$.

Example 5.10: Let functional properties be described in terms of tolerances as specified in Example 5.6. Let the single functional property *low power consumption* be described by the interval [45, 65]. Then the *closure* of [45, 65] is a collection of “nested” closed intervals; i.e., a collection $\{S_n\}$ of intervals such that for each interval S_n , $S_n \subset [45, 65]$. This means that if the observed power consumption of an artifact satisfies S_n , then its displayed power consumption lies within the tolerance [45, 65].

The next definition relates to designing from a set of real artifacts in a catalogue.

Definition 5.10 (Closure in Function Space Catalogue, $U_{\mathcal{F}}$): Let f be a functional property (or a design specification) in \mathcal{F} , and \mathcal{S} be the set of all real artifacts that existed in the past, and exist presently in a catalogue. Then $g \in U_{\mathcal{F}}(f)$ if and only if the set of artifacts in catalogue \mathcal{S} that satisfy the functional property g is a subset of the set of artifacts in catalogue \mathcal{S} that satisfy the functional property f . Each element in $U_{\mathcal{F}}(f)$ represents an *alternative* functional description for f . If F is a set of functional properties in \mathcal{F} , then $U_{\mathcal{F}}(F) = \bigcup_{f \in F} U_{\mathcal{F}}(f)$ (in which case $U_{\mathcal{F}}$ is called *additive*).

If $U_{\mathcal{F}}(f)$ designates the set of *all* functional descriptions of artifacts in catalogue \mathcal{S} that qualify the functional property f , then $U_{\mathcal{F}}(f)$ is called a *topological closure in the function space catalogue*. The following conditions are satisfied by a topological closure in the function space catalogue:

1. $U_{\mathcal{F}}(\emptyset) = \emptyset$;
2. $F \subset U_{\mathcal{F}}(F)$ for each $F \subset \mathcal{F}$;
3. $U_{\mathcal{F}}(F \cup G) = U_{\mathcal{F}}(F) \cup U_{\mathcal{F}}(G)$ for each $F \subset \mathcal{F}$ and $G \subset \mathcal{F}$;
4. $U_{\mathcal{F}}(U_{\mathcal{F}}(F)) = U_{\mathcal{F}}(F)$ for each $F \subset \mathcal{F}$.

Example 5.11: If f is the functional property *ease of achieving 105 - 125 DbA \wedge ease of achieving 2000 - 5000 Hz \wedge resistance to temperature*, then $U_{\mathcal{F}}(f)$ includes $f_1 = ease\ of\ achieving\ 105 - 125\ DbA \wedge ease\ of\ achieving\ 2000 - 5000\ Hz \wedge resistance\ to\ temperature \wedge high\ response\ time$ since the car horn A is the only artifact in the catalogue \mathcal{S} that satisfies both f and f_1 .

The next definition formalizes the intuitive notion of proximity in the attribute space.

Definition 5.11 (Closure Attribute Space): If $U_{\mathcal{D}}$ is a single-valued relation on $2^{\mathcal{D}}$ with a range of $2^{\mathcal{D}}$, then we say that $U_{\mathcal{D}}$ is a *closure operation* (or “closure”) for \mathcal{D} provided that the following conditions are satisfied:

1. $U_{\mathcal{D}}(\emptyset) = \emptyset$;
2. $M \subset U_{\mathcal{D}}(M)$ for each $M \subset \mathcal{D}$;
3. $U_{\mathcal{D}}(M \cup N) = U_{\mathcal{D}}(M) \cup U_{\mathcal{D}}(N)$ for each $M \subset \mathcal{D}$ and $N \subset \mathcal{D}$;

If m is a structural property in \mathcal{D} , then $U_{\mathcal{D}}(m)$ designates a set of structural descriptions in \mathcal{D} such that each structural description qualifies (derives) by means of *logical inference* the structural property m . Each element in the set represents a different “viewpoint” of the attribute m . If M is a set of attributes in \mathcal{D} , then $U_{\mathcal{D}}(M) = \bigcup_{m \in M} U_{\mathcal{D}}(m)$ (in which case $U_{\mathcal{D}}$ is called *additive*).

The structure $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is called a *closure attribute space*. In the closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$, the attribute x is *proximal* to M if and only if $x \in U_{\mathcal{D}}(M)$.

Definition 5.12 (Topological Attribute Space): A *topological attribute operation* (or “topological closure”) for a set \mathcal{D} is a closure operation $U_{\mathcal{D}}$ for \mathcal{D} satisfying the following condition: $F \subseteq \mathcal{D}$ implies $U_{\mathcal{D}}(U_{\mathcal{D}}(F)) = U_{\mathcal{D}}(F)$. $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is called a *topological attribute space*.

If $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is a topological attribute space, then for every structural property m , $U_{\mathcal{D}}(m)$ designates the set of *all* structural descriptions in \mathcal{D} that qualify (by means of *logical inference*) the attribute m .

Example 5.12: Let \mathcal{D} be the set of all artifact descriptions of fasteners obtained by conjunction (\wedge), disjunction (\vee), and negation (\sim) over the basic attributes in Table 5.1. If the set M contains the single attribute *polycarbonate material*, then the *closure* of M in $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ or under $U_{\mathcal{D}}$ contains, among others, the following proximal attributes (all of them derive *polycarbonate material*): (1) *polycarbonate material*; (2) *polycarbonate material* \wedge *fastener with washer*; (3) *polycarbonate material* \wedge \sim *chamfer point*; and (4) *polycarbonate material* \wedge (*threaded shank* \vee *threaded tail*) \wedge (*cap drive recess* \vee *external socket hex*). Each proximal attribute represents an alternative “viewpoint” of *polycarbonate material*.

The next definition relates to designing from a set of real artifacts in a catalogue.

Definition 5.13 (Closure in Attribute Space Catalogue, $U_{\mathcal{D}}$): Let m be a structural property in \mathcal{D} , and \mathcal{S} be the set of all real artifacts that existed in the past, and exist presently in a catalogue. Then $l \in U_{\mathcal{D}}(m)$ if and only if the set of artifacts in the catalogue \mathcal{S} that have the attribute l is a subset of the set of artifacts that have the attribute m . Each element in the set represents an *alternative* structural description. If M is a set of attributes in \mathcal{D} , then $U_{\mathcal{D}}(M) = \bigcup_{m \in M} U_{\mathcal{D}}(m)$ (in which case $U_{\mathcal{D}}$ is called *additive*).

If $U_{\mathcal{D}}(m)$ designates the set of *all* structural descriptions of artifacts in catalogue \mathcal{S} that qualify the attribute m , then $U_{\mathcal{D}}(m)$ is called a *topological closure in attribute space catalogue*. The following conditions are satisfied by a topological closure in the attribute space catalogue:

1. $U_{\mathcal{D}}(\emptyset) = \emptyset$;
2. $M \subset U_{\mathcal{D}}(M)$ for each $M \subset \mathcal{D}$;
3. $U_{\mathcal{D}}(M \cup N) = U_{\mathcal{D}}(M) \cup U_{\mathcal{D}}(N)$ for each $M \subset \mathcal{D}$ and $N \subset \mathcal{D}$;
4. $U_{\mathcal{D}}(U_{\mathcal{D}}(M)) = U_{\mathcal{D}}(M)$ for each $M \subset \mathcal{D}$.

Example 5.13: If m is the attribute *shoulder \wedge threaded shank \wedge truss head type*, then $U_{\mathcal{D}}(m)$ includes the structural description $l = \textit{shoulder \wedge truss head type}$ since fastener E is the only fastener that satisfies both attributes m and l .

Definition 5.14 (Open and Closed Sets): A subset F of a closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is called *closed* if $U_{\mathcal{F}}(F) = F$, and *open* if its complement (relative to \mathcal{F}) is closed; i.e., if $U_{\mathcal{F}}(\mathcal{F} - F) = \mathcal{F} - F$. Similarly, a subset M of a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is called *closed* if $U_{\mathcal{D}}(M) = M$, and *open* if its complement (relative to \mathcal{D}) is closed; i.e., if $U_{\mathcal{D}}(\mathcal{D} - M) = \mathcal{D} - M$. Thus, closed sets of a closure function space (closure attribute space) are the “fixed elements” of $U_{\mathcal{F}}(U_{\mathcal{D}})$.

Example 5.14: For a topological function space (see Definition 5.9) the closure of each subset of \mathcal{F} is closed. If f is a functional property in \mathcal{F} , then the closed set $U_{\mathcal{F}}(f)$ designates the set of all functional descriptions that derive the functional property f . A functional property that is not included in $U_{\mathcal{F}}(f)$ does not derive f . The set of all functional descriptions that do not derive f is the complement of $U_{\mathcal{F}}(f)$ relative to \mathcal{F} . Therefore, the closure of the complement of $U_{\mathcal{F}}(f)$ includes the set of all functional descriptions that do not derive f . Thus, in accordance with Definition 5.14, $U_{\mathcal{F}}(f)$ is open. In summary, every closure in $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is closed and open at the same time. An analogous result is obtained for topological

attribute spaces.

Definition 5.15 (Limit Point of a Set): A *limit point* of a set F (or M) in a closure function space (or closure attribute space) is a point x belonging to the closure of $F - \{x\}$ (or $M - \{x\}$). The set of all limit points of a set F (or M) is denoted by F' (or M') and called the *derivative* of F (or M) in the closure function space (or closure attribute space). Clearly, the closure of a set F (or M) is the union of the F (or M) with its set of limit points.

Example 5.15: If f is the functional property *low power consumption* and $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is a topological function space, then *low power consumption* \wedge *ease of maintenance* is a limit point of f since $\{\text{low power consumption} \wedge \text{ease of maintenance}\} \in U_{\mathcal{F}}(\{\text{low power consumption}\})$.

Definition 5.16 (Neighborhood): A *neighborhood* of a functional property f is any closure $U_{\mathcal{F}}(F)$ for some set F such that $f \in U_{\mathcal{F}}(F)$. Similarly, a *neighborhood* of a structural property m is any closure $U_{\mathcal{D}}(M)$ for some set M such that $m \in U_{\mathcal{D}}(M)$.

Example 5.16: If f is the functional property *low power consumption* \wedge *ease of maintenance* and $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is a topological function space, then $U_{\mathcal{F}}(\{\text{low power consumption}\})$ and $U_{\mathcal{F}}(\{\text{ease of maintenance}\})$ are two distinct neighborhoods of f . This means that f implies two distinct functional properties in \mathcal{F} .

Definition 5.17 (Convergence): Let $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ be a closure function space and $\{f_n\}_{n=1}^{\infty}$ be a sequence of functional properties of \mathcal{F} . Then $\{f_n\}_{n=1}^{\infty}$ *converges* to the functional property $f \in \mathcal{F}$ (f is a *limit* of the sequence), if for every neighborhood $U_{\mathcal{F}}$ of f there is a positive integer N such that $f_n \in U_{\mathcal{F}}$ for all $n \geq N$. Similarly, let $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ be a closure attribute space and $\{m_n\}_{n=1}^{\infty}$ be a sequence of attributes of \mathcal{D} . Then $\{m_n\}_{n=1}^{\infty}$ *converges* to the structural property $m \in \mathcal{D}$ (or m is a *limit* of the sequence), if for every neighborhood $U_{\mathcal{D}}$ of m there is a positive integer N such that $m_n \in U_{\mathcal{D}}$ for all $n \geq N$.

Example 5.17: Let functional properties be described in terms of tolerances (determined by intervals) as specified in Example 5.6. If the functional property *low power consumption* is described by intervals, and Cantor's Nested Intervals Theorem [5] is used; then any sequence of "shrinking" or "nested" closed intervals with diameters approaching zero must have exactly one value of power consumption in common.

Example 5.18: If we evolve an artifact structure by utilizing conjunctions of

attributes, we get an attribute as the limit of the evolution. For example, the following sequence converges to the structural description of fastener F:

$m_1 = \text{slot drive recess};$

$m_2 = \text{slot drive recess} \wedge \text{round head type};$

$m_3 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder};$

$m_4 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder} \wedge \text{threaded shank};$

$m_5 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder} \wedge \text{threaded shank} \wedge \text{threaded tail};$

$m_6 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder} \wedge \text{threaded shank} \wedge \text{threaded tail} \wedge \text{chamfer point};$

$m_7 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder} \wedge \text{threaded shank} \wedge \text{threaded tail} \wedge \text{chamfer point} \wedge \sim \text{washer};$

$m_8 = \text{slot drive recess} \wedge \text{round head type} \wedge \text{shoulder} \wedge \text{threaded shank} \wedge \text{threaded tail} \wedge \text{chamfer point} \wedge \sim \text{washer} \wedge \sim \text{polycarbonate material}.$

Note that the limit here is an element of \mathcal{D} ; therefore, it may only be an approximation of the design solution F. This corresponds to real design, where artifacts are described by a finite number of properties, and thus each implicitly represents an infinite number of possible designs.

5.3.3 TRANSFORMATION BETWEEN FUNCTION AND ATTRIBUTE SPACES

The automation of design will be improved by providing a rule of direct correspondence between the function space and the attribute space without the restriction of a catalogue. For example, in designing a helical compression spring subject to a known load, maximum allowable stress, shear modulus, safety factor, deflection, and free length; the designer applies a known formula to obtain the dimensions of the helical spring (such as the coil radius, wire diameter, number of coils, and spring index) that correspond to the helical spring specification. Often, there is a set of production rules (besides numerical equations) that produce a design solution in terms of attributes by giving a specification in terms of functional properties. *Analysis* is concerned with the process of inferring potential functionality from an artifact structure. In contrast, designing is mainly concerned with *synthesis*: the generation of an artifact structure that will satisfy a desired function. The mapping from the function to the artifact description (i.e., synthesis) is often the “inverse” of the mapping from the artifact description to the function (i.e., analysis). The following two definitions formalize these intuitive definitions of synthesis and analysis as described in Figure 5.1.

In an “ideal” artifact, the number of structural properties is equal to the number of functional properties. When this is the case, a perturbation in a particular structural property must affect only its referent functional property. In general, there are two categories of artifacts: *uncoupled*, and *coupled* [13]. An uncoupled artifact can be an

“ideal” artifact, whereas a coupled artifact renders some of the functional properties dependent on other functional properties. A coupled artifact may result when there are less attributes than functional properties. Thus, a coupled artifact may be decoupled by adding extra components, which increases the number of structural properties. In some special cases, redundancy may be required to increase the reliability of a particular subsystem, which also increases the safety of the entire system. A redundant artifact may result when there are more attributes than functional properties.

If each artifact in a particular design domain (e.g., the fasteners domain) is uncoupled, the total number of structural properties in the domain is equal to the total number of functional properties in the domain. In other words, the cardinality of the function space is equal to the cardinality of the attribute space. Since this equality is unlikely, many design domains exhibit the deficiency of having more functions than attributes. When this is the case, the synthesis mapping is often unable to discriminate between the functionality of some structural descriptions.

Definition 5.18 (Analysis mapping): If Γ is a single-valued relation on \mathcal{D} with a range of \mathcal{F} , then we say that Γ is an *analysis mapping* from the artifact description to the functional description provided that the following conditions are satisfied: If m is a structural property in \mathcal{D} , then $\Gamma(m)$ designates a functional description in \mathcal{F} that corresponds to the structural property m . If M is a set of structural properties in \mathcal{D} , then $\Gamma(M) = \bigcup_{m \in M} \Gamma(m)$. Analysis mapping is concerned with inferring potential functionality from an artifact structure.

Example 5.19: If M contains the single structural property *polycarbonate material*, then $\Gamma(M)$ can be generated in the following way: We start with {A, B, C, D, G} as the *polycarbonate material* designs as summarized in Table 5.1. The shared functional properties (see Table 5.2) of the *polycarbonate material* fasteners are {*corrosion resistance & thermal conductivity*}. Therefore, we may say that $\Gamma(\{\textit{polycarbonate material}\}) = \{\textit{corrosion resistance \& thermal conductivity}\}$.

Example 5.20: Let functional and structural properties be described in terms of tolerances (determined by intervals) as specified in Example 5.6. Let the structural attribute *polycarbonate material* denote the percent of carbon that exists in the material. If the percent of *polycarbonate material* is described by the tolerance $[a, b]$, then it is mapped to a tolerance of the functional property *corrosion resistance & thermal conductivity* by the following analysis mapping: $\Gamma([a, b]) = \{[a^2, b^2]\}$.

Definition 5.19 (Synthesis Mapping): If Υ is a single-valued relation on \mathcal{F} with a range of \mathcal{D} , then we say that Υ is a *synthesis mapping* from the functional description to the artifact description provided that the following conditions are satisfied: If f is a functional property in \mathcal{F} , then $\Upsilon(f)$ designates an attribute in \mathcal{D} that corresponds to the functional property f . If F is a set of functional properties in \mathcal{F}

then $Y(F) = \bigcup_{f \in F} Y(f)$. Synthesis mapping is concerned with the generation of an artifact structure that will satisfy a desired function.

Example 5.21: If F contains the single functional property *high strength*, then $Y(F)$ can be generated in the following way: We start with $\{B, D, E, F\}$ as the *high strength* designs as summarized in Table 5.2. The shared structural attributes (see Table 5.1) of the *high strength* fasteners are $\{\textit{shoulder, threaded shank, threaded tail, chamfer point}\}$. Therefore, we may say that $Y(\{\textit{high strength}\}) = \{\textit{shoulder} \wedge \textit{threaded shank} \wedge \textit{threaded tail} \wedge \textit{chamfer point}\}$.

Example 5.22: Additional synthesis mapping can be defined; e.g., $Y(\{\textit{high strength}\})$ may designate a structural description of an artifact in catalogue \mathcal{S} that derives the functional property *high strength*.

Often the mapping from the function to the artifact description (i.e., synthesis) is the “inverse” of the mapping from the artifact description to the function (i.e., analysis). This can be formalized as $Y \equiv \Gamma^{-1}$; i.e., $\forall f \in \mathcal{F}, Y(f) = \{m \in \mathcal{D} : \Gamma(m) = f\}$.

The idealized design process cycle is summarized in the following definition:

Definition 5.20 (Idealized Design Process): The structure $\langle \langle \mathcal{F}, U_{\mathcal{F}} \rangle, \langle \mathcal{D}, U_{\mathcal{D}} \rangle, Y, \Gamma \rangle$ is called an *idealized design process*. The idealized design process cycle is depicted in Figure 5.6.

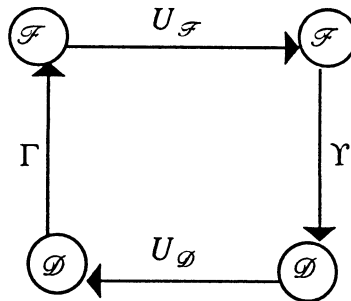


Figure 5.6 The Idealized Design Process Cycle

The next example illustrates the intuitive definition of an idealized design process (see Figure 5.2) as the designation of a domain in the attribute space \mathcal{D} that corresponds to a specification domain in the function space \mathcal{F} .

Example 5.23 (An Idealized Design Process): If F contains the single functional property *high strength*, then the closure in function space $U_{\mathcal{F}}(\{\textit{high strength}\})$ generates the proximal functional description $\{\textit{high strength} \wedge \textit{retractability}\}$, which

derives the functional property *high strength*. Following the procedure suggested in Example 5.21, $Y(\{high\ strength \wedge retractability\})$ generates the set of structural descriptions that are shared by fasteners {B, D, E, F}; i.e., $\{shoulder \wedge threaded\ shank \wedge threaded\ tail \wedge chamfer\ point\}$. Then, the closure in attribute space $U_{\mathcal{D}}(\{shoulder \wedge threaded\ shank \wedge threaded\ tail \wedge chamfer\ point\})$ produces the structural description of fastener B. To recap, in the specification *high strength*, B (or its structural description) is a design solution.

5.3.4 DECOMPOSITION OF DESIGN SPECIFICATION

The overall functions (design specifications) that an artifact has to satisfy depend on the design domain. Design functions may include information that is relatively abstract. Also, some functions may qualify (or derive) other functions. Therefore, in the function space, each function selected for a given specification can be further decomposed into sub-functions so that enough information is provided to proceed to the next design stages. Functions are decomposed until it is possible to match them with attributes at the synthesis stage. Decomposition of functions can be done by a designer, an intelligent advisor, or both. A designer performs decomposition according to experience and design knowledge. An intelligent advisor is a knowledge-based system supported by catalogues containing the principles of functional decomposition. Given the function space, a number of alternative function decompositions may exist; each of which satisfies the overall design specification. Such decomposition is represented in the function space and used in the final stage of conceptual design, namely synthesis.

The number of function levels depends on the complexity of the designed artifact. The logical hierarchy of functions presents a clear view of the designed artifact and defines its behavior, which means that the lower function levels are viewed as interacting to produce the overall functions. The lower function levels represent design problems of reduced complexity and with lower levels of abstraction. This interaction may have various logical forms (e.g., conjunction, disjunction, etc.). The set of functions, sub-functions, and the logical relationships between them provide a description of the function space for a class of artifacts that can be designed.

A highly articulated version of the above view of the design process is formulated in the literature as the analysis-synthesis-evaluation (ASE) paradigm (see Chapter 2). According to the ASE paradigm, the design process consists of three logically and temporally distinct stages: decomposition of functions, synthesis, and evaluation.

The analysis and synthesis stages are embedded in the idealized design process framework as follows: Given the overall design specification, $f \in \mathcal{F}$ the designer applies the closure operation $U_{\mathcal{F}}$ (in function space \mathcal{F}) several times. Each single application of $U_{\mathcal{F}}$ generates alternative function decompositions. This stage is followed by *synthesis mapping* Y from the function space to the artifact description.

This results in a number of alternative design solutions that are given in terms of their structural descriptions.

Figure 5.7 illustrates *alternative solution paths*, each of which satisfies the overall design specification f_0 . Given the overall design specification f_0 , the first decomposition stage generates three alternative functional descriptions $U_{\mathcal{F}}(\{f_0\}) = \{f_0, f_1, f_2, f_3\}$, followed by a second stage of decomposition $U_{\mathcal{F}}(\{f_0, f_1, f_2, f_3\}) = \{f_0, f_1, f_4, f_5, f_3\}$. At this point, it is possible to match the functions with attributes. A stage of synthesis follows where four alternative artifact descriptions are determined $\{m_1, m_2, m_3, m_4\}$.

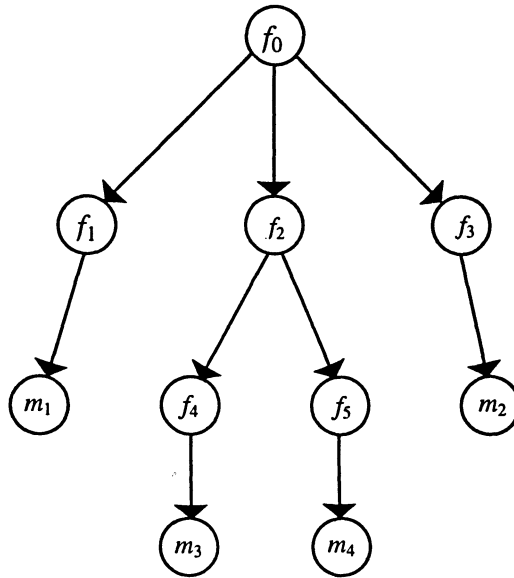


Figure 5.7 Decomposition of Functions and the Corresponding Synthesis Stage

Example 5.24: consider the fastener domain where the initial specification is $f_0 = \text{high strength} \wedge \text{high retractability} \wedge \text{medium precision}$. The first iteration may be to realize that strength is characterized separately by the head and tail of the fastener. By applying the knowledge-base of fasteners and analyzing the current requirements, it is concluded that both of them demand high strength; i.e., $U_{\mathcal{F}}(\{f_0\}) = f_1 = \{\text{high head strength} \wedge \text{high tail strength} \wedge \text{high retractability} \wedge \text{medium precision}\}$. The second iteration may be to get a better definition of retractability. There is a choice of how fasteners are driven and in order to get high retractability we need rotary-mode fasteners; i.e., $U_{\mathcal{F}}(\{f_1\}) = f_2 = \{\text{high head strength} \wedge \text{high tail strength} \wedge \text{drive-mode rotary} \wedge \text{tail-mode rotary} \wedge \text{medium precision}\}$. In the next

iteration, we realize that medium precision translates into a looser body fit; i.e., $U_{\mathcal{F}}(\{f_2\}) = f_3 = \{high\ head\ strength \wedge high\ tail\ strength \wedge drive\text{-}mode\ rotary \wedge tail\text{-}mode\ rotary \wedge looser\ body\ fit\}$. At this point, synthesis decisions begin to be made as the final configuration of the design begins to take place. From the current information, we infer that the head of the fastener needs a shoulder, the tail of the fastener will be fastened using threads, the drive type will be a hex head, and the body will have threads. Finally, it is decided that the tip of the fastener should be chamfered in order to maximize reversibility. Thus, the overall synthesis process can be formulated as: $\Upsilon(f_3) = \{shoulder\ head\ type \wedge threaded\ tail\ type \wedge socket\text{-}hex\ drive\ type \wedge threaded\ body\ type \wedge tip\ type\ chamfer\}$.

5.3.5 CONVERGENCE OF THE FUNCTION DECOMPOSITION STAGE

According to the ASE paradigm, synthesis mapping can not be applied unless the initial design specification can not be further decomposed. In mathematical terms, it means that for some decomposition stage i and set of functions F^i , $U_{\mathcal{F}}(F^i) = F^i$. A subset F of a function space \mathcal{F} will be called *closed* if $U_{\mathcal{F}}(F) = F$. Closed sets of a structure $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ are the “fixed points” of $U_{\mathcal{F}}$. Thus, the existence of fixed points for the operator $U_{\mathcal{F}}$ is a necessary condition for the design process to terminate with a solution that satisfies the initial specification. The next result establishes a condition under which fixed points for $U_{\mathcal{F}}$ exist.

Theorem 5.1: Let U be a single-valued relation on $2^{\mathcal{F}}$ with a range of $2^{\mathcal{F}}$. If U is monotonous; i.e., it satisfies the property $F_1 \subseteq F_2 \Rightarrow U(F_1) \subseteq U(F_2)$, then U has a fix-point in $2^{\mathcal{F}}$.

Since the closure operation in function space $U_{\mathcal{F}}$ is additive, it is also monotonous. Therefore, Theorem 5.1 guarantees the existence of a fixed-point for $U_{\mathcal{F}}$.

5.3.6 ORDER RELATION FOR ATTRIBUTE AND FUNCTION SPACES

If a space is metric; that is, one can calculate a *distance* between any two entities in the function and attribute spaces, then the distance measure can be used to assign *values* to each of the attributes or functions describing an artifact such as those described in Tables 5.1 – 5.3. There are no clear performance metrics for functions. Evaluation of structural attributes is straightforward. Performance metrics include, but are not limited to: size, weight, power requirements, efficiency, capacity for force generation, and economic features. Unfortunately, precise performance metrics are difficult to articulate for designs that are completely described as function structures.

However, the hierarchical design record provided by designing on each successive level of abstraction allows the designer to monitor the effectiveness of individual functional entities. This design approach begins to address the fundamental problem of articulating performance metrics for function structures by imposing a clear association between the functions and the derived structural attributes.

A simple example of the benefits of a metric is related to the search for alternative design solutions. A logic tree is one of the tools that can assist the designer to represent and record the function decomposition stage. A search problem is characterized by an initial state, a set of operators, and a goal state [6]. For example, the overall specification is the initial state, a set of production rules for the decomposition is the set of operators, and the design solution is the goal state. In design, the initial state is known *a priori*, while the goal state is to be determined. An operator transforms one state to another. The objective is to find a sequence of operators that will lead to a goal state. Starting from the initial state, the set of all states that can be reached by applying operators is called the *state space*. In order to generate a solution, an extensive search of the state space may be required. Three representative methods [7] of guiding this search are: (1) breadth-first, (2) depth-first, and (3) best-first. One of the most promising best-search methods is the A^* method [7]. At each step of the A^* search process, the most promising state (e.g., state with the lowest corresponding evaluation function cost), which has been generated but not further expanded, is selected. If it happens to be the final solution, then the search process terminates. If not, the chosen state is expanded by generating its successive (new) states. A heuristic state evaluation function is applied to each successive state to compute a value. Then, all the new states are added to the list of unexpanded states. Again the most promising state is selected and the process described above continues. The heuristic state evaluation function in the A^* algorithm has a quantitative nature. That is, the associated cost of each state should allow for ordinal comparison among states; for example, functional property 2 is better than functional property 5 because the value of the evaluation of functional property 2 is less than that of functional property 5. The next definition pertains to assigning *values* to each of the entities in the function and attribute spaces such as those described in Tables 5.1 – 5.3.

Definition 5.21 (Designer Preferences in Function Space): A binary relation of “preference or indifferent” $\succ_{\mathcal{F}}$ over \mathcal{F} satisfying:

1. If f_1, f_2 are members of \mathcal{F} , then either $f_1 \succ_{\mathcal{F}} f_2$ or $f_2 \succ_{\mathcal{F}} f_1$. It is interpreted as “the designer strictly prefers one of or is indifferent to the functions f_1 or f_2 relative to a performance metric for the function space.”
2. If $f_1 \succ_{\mathcal{F}} f_2$, then $f_2 \succ_{\mathcal{F}} f_1$ is false;
3. If $f_1 \succ_{\mathcal{F}} f_2$ and $f_2 \succ_{\mathcal{F}} f_3$, then $f_1 \succ_{\mathcal{F}} f_3$.

Designer preference in attribute space (denoted by $\succ_{\mathcal{O}}$) is defined in a similar manner.

The following generalizes the previous definition to account for preferences over sets of functions.

Definition 5.22 (Preference over Sets of Functions): A binary relation of “preference or indifferent” $\succ_{\mathcal{F}}$ on $2^{\mathcal{F}}$ satisfying: If F_1, F_2 are sets of $2^{\mathcal{F}}$, then $F_1 \succ_{\mathcal{F}} F_2$ if and only if for any function $f_1 \in F_1$ and any function $f_2 \in F_2$, $f_1 \succ_{\mathcal{F}} f_2$. This relation is not necessarily defined for any two subsets of $2^{\mathcal{F}}$. Preference over sets of structural attributes is defined in a similar manner.

5.4 IDEALIZED DESIGN PROCESS AXIOMS

5.4.1 CONTINUITY IN FUNCTION AND ATTRIBUTE SPACES

Idealized design process properties (or axioms) convey the assumptions of the theory about the nature of artifacts and their potential manipulation to achieve a desired functionality. These axioms are the foundations of the theorems discussed later. Some of these axioms can be excluded thus providing different categorizing of idealized design processes. This enables an assessment of their relevance to design. The main purpose of the remaining sections in this chapter is to investigate the notion of the continuous mapping of one closure space to another. We visualize the continuity of synthesis mapping (or analysis mapping) at f (or m) by thinking that whenever f is “close” to a set F , then $Y(f)$ is “close” to $Y(F)$.

Definition 5.23 (Continuous Synthesis Mapping, CS): A synthesis mapping of a closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ to a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is said to be *continuous at a functional property f of \mathcal{F}* if $F \subseteq \mathcal{F}; f \in U_{\mathcal{F}}(F)$ implies $Y(f) \in U_{\mathcal{D}}(Y(F))$. The mapping Y is said to be *continuous* if it is continuous at each functional property f of \mathcal{F} ; or likewise, if $F \subseteq \mathcal{F}$ implies $Y(U_{\mathcal{F}}(F)) \subseteq U_{\mathcal{D}}(Y(F))$.

A closure operation for a space P was defined as a single-valued relation on 2^P with a range of 2^P . The intuitive meaning of a closure for a set (in function and/or attribute spaces) determines, roughly speaking, what points are proximal to which sets. This is the intuitive sense of the notion of a closure operation. Thus, the definition of continuous mapping as a mapping that preserves the relation $\{f \text{ is proximal to } F\}$ becomes evident: $\{f \text{ is proximal to } F\}$ implies the relation $\{Y(f) \text{ is proximal to } Y(F)\}$. The continuity property of a synthesis mapping guarantees that a small change in the artifact functionality will result in a small change in the artifact description. Therefore, if the current candidate’s functionality differs slightly from the required function, a small modification to the structure may be sufficient. Convergence is also a process-oriented concept (see Definition 5.17), which provides

a different perspective of continuity. Convergence guarantees that a sequence of incremental refinements to the artifact functionality will cause only small incremental changes to the structure.

Example 5.25: Assume that fastener F has the property *corrosion resistance & thermal conductivity*. Given the initial specification *corrosion resistance & thermal conductivity*, $U_{\mathcal{F}}(\{ \text{corrosion resistance \& thermal conductivity} \})$ - the closure in function space - generates the proximal functional description *corrosion resistance & thermal conductivity* \wedge *medium precision*. This functional description is mapped, by the synthesis mapping, to the structural description of fastener F.

On the other hand, since the structural property *polycarbonate material* is the most probable structural attribute that derives the functional property *corrosion resistance & thermal conductivity*, the designer may decide to synthesize a fastener that is composed of *polycarbonate material*. The closure operation in attribute space ($U_{\mathcal{D}}$) generates the set of structural descriptions that correspond to fasteners A, B, C, D, and G; each of which is made up of polycarbonate material. Thus, the synthesis mapping is not continuous at the functional property *corrosion resistance & thermal conductivity* since F is not included in the set {A, B, C, D, G}.

Definition 5.24 (Continuous Analysis Mapping, CA): An analysis mapping of a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ to a closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is said to be *continuous at a structural property m of \mathcal{D}* if $M \subseteq \mathcal{D}$; $m \in U_{\mathcal{D}}(M)$ implies $\Gamma(m) \in U_{\mathcal{F}}(\Gamma(M))$. The mapping Γ is said to be *continuous* if it is continuous at each structural property m of \mathcal{D} ; or similarly, if $M \subseteq \mathcal{D}$ implies $\Gamma(U_{\mathcal{D}}(M)) \subseteq U_{\mathcal{F}}(\Gamma(M))$.

The continuity property of an analysis mapping guarantees that a small change in the artifact description will result in a small change in the artifact functionality. Therefore, if the current candidate's structure differs slightly from the required structure, a small modification to the functionality may be sufficient. Furthermore, it may occur that the analysis mapping is discontinuous at the structural description of the artifact; thus, cannot be analyzed. Therefore, the artifact cannot be synthesized to a description close to the discontinuity point. The continuity property of an analysis mapping also guarantees that a sequence of incremental refinements to an artifact structure will cause only small incremental changes to the functionality. In other words, each convergent sequence of structures in the attribute space will cause the convergence of the corresponding manifestations of functionality in the function space.

As mentioned above, the synthesis mapping from the function to the artifact description is often the "inverse" of the analysis mapping from the artifact description to the function. This can be formalized as $\Upsilon \equiv \Gamma^{-1}$; i.e., $\forall f \in \mathcal{F}, \Upsilon(f) = \{m \in \mathcal{D}: \Gamma(m) = f\}$. Furthermore, the analysis mapping is often one-to-one, which means that distinct structural attributes in \mathcal{D} are mapped to distinct functional attributes in \mathcal{F} ,

and thus manifest different functions. Closure spaces $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ and $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ are *topologically equivalent* or *homeomorphic* if there is a one-to-one analysis mapping Γ from the attribute space to the function space in which Γ and the inverse function Γ^{-1} (i.e., Υ) are continuous; i.e., they satisfy the CA and CS properties respectively. This type of analysis mapping Γ is called a *homeomorphism*.

We note that the following condition is not sufficient for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ and $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ to be *homeomorphic*: there exists a one-to-one continuous analysis mapping Γ of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, and a one-to-one continuous synthesis mapping Υ (*distinct* from Γ^{-1}) of $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ onto $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$. The notion of homeomorphism is fundamental and therefore we provide a necessary and sufficient condition for an analysis mapping to be a homeomorphism.

Theorem 5.2: Let the analysis mapping Γ be a one-to-one mapping of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, and the synthesis mapping be the inverse of Γ (i.e., $\Upsilon \equiv \Gamma^{-1}$); then Γ is a *homeomorphism* if the following condition holds: $F \subseteq \mathcal{F}$ implies $\Upsilon(U_{\mathcal{F}}(F)) = U_{\mathcal{D}}(\Upsilon(F))$.

Definition 5.25 (Order Preserving under Synthesis, OPS): A synthesis mapping of a closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ onto a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ “preserves order” if and only if the binary relations $\succ_{\mathcal{F}}$ and $\succ_{\mathcal{D}}$ satisfy: $F_1 \succ_{\mathcal{F}} F_2$ if and only if $\Upsilon(F_1) \succ_{\mathcal{D}} \Upsilon(F_2)$.

The order preserving property means that the preference relation between two functional properties is preserved under the synthesis mapping. This property is related to the fundamental problem of articulating performance metrics for function structures by imposing a clear association between the functions and the derived structural attributes.

Definition 5.26 (Order Preserving in Function Space, OPFS): A closure operation in a closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is said to be order preserving if and only if $F_1 \succ_{\mathcal{F}} F_2$ then $\neg[U_{\mathcal{F}}(F_2) \succ_{\mathcal{F}} U_{\mathcal{F}}(F_1)]$.

This property means that if the designer prefers (or is indifferent to) the set of functional properties F_1 to the set of functional properties F_2 (relative to a performance metric for function space), then at least one proximal description of F_2 performs worse than another proximal solution of F_1 .

Similarly, we define “order preserving in attribute space”:

Definition 5.27 (Order Preserving in Attribute Space, OPAS): A closure

operation in a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is said to be order preserving if $M_1 \succ_{\mathcal{D}} M_2$ then $\neg[U_{\mathcal{D}}(M_2) \succ_{\mathcal{D}} U_{\mathcal{D}}(M_1)]$.

This property means that if the designer prefers (or is indifferent to) the set of structural properties M_1 to the set of functional properties M_2 (relative to a performance metric for attribute space), then at least one proximal description of M_2 performs worse than another proximal solution of M_1 .

The idealized design process axioms are interrelated as follows:

Theorem 5.3 (Consistency): Let the analysis mapping Γ be a one-to-one mapping of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, the synthesis mapping be the inverse of Γ (i.e., $\Upsilon \equiv \Gamma^{-1}$), and Γ be a *homeomorphism*. Then,

1. The OPS & OPAS imply the OPFS;
2. The OPS & OPFS imply the OPAS.

A property P of closure function and attribute spaces is a *topological property* or *topological invariant* provided that if closure function space $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ has property P , then so does every closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ that is *homeomorphic* to $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$. For example, “the cardinal number of \mathcal{F} is m ” is a topological property. Next, we show that the property “ F is a fixed point in $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ ” is a topological invariant.

Theorem 5.4: Let the analysis mapping Γ be a one-to-one mapping of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, the synthesis mapping be the inverse of Γ (i.e., $\Upsilon \equiv \Gamma^{-1}$), and Γ be a *homeomorphism*. If $U_{\mathcal{F}}$ is monotonous (and thus has a fix-point in $2^{\mathcal{F}}$ according to Theorem 5.1), then $U_{\mathcal{D}}$ is monotonous (and thus has a fixed point in $2^{\mathcal{D}}$ according to Theorem 5.1).

Theorem 5.5: Let the analysis mapping Γ be a one-to-one mapping of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, the synthesis mapping be the inverse of Γ (i.e., $\Upsilon \equiv \Gamma^{-1}$), and Γ be a *homeomorphism*. If F^* is a fixed point of $U_{\mathcal{F}}$, then $\Upsilon(F^*)$ is a fixed point of $U_{\mathcal{D}}$.

5.4.2 THE COMPLEXITY OF THE “HOMEOMORPHISM” AND “CONTINUITY” PROBLEMS

Even though the concept of homeomorphism is useful in design, it is often difficult to discover whether closure function and attribute spaces are homeomorphic or not. Likewise deciding whether a synthesis mapping of $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, onto $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ is continuous at some functional property is a difficult problem. In general (i.e., for instances of any size), the *Homeomorphism* and *Continuity* problems are *undecidable*. That is, a problem is undecidable if there is no algorithm that takes an instance of the problem as input and determines whether the answer to that instance is “yes” or “no.” The purpose of this section is to investigate the worst-case computational complexity of the Homeomorphism and Continuity problems.

Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources (such as computation time, space and hardware size), which are required to solve problems through digital algorithms. A *problem* is a collection of *instances* that share a mathematical form, but differ in size and in the values of numerical constants. In general, we convert optimization problems to *decision problems* by posing the question of whether there is a feasible solution to a given problem, which has an objective function value that is equal to or greater than a specified threshold. The notion of “easy to verify” but “not necessarily easy to solve” is at the heart of the Class *NP* decision problems. Specifically, NP problems include all the decision problems that could be polynomial-time solved if the right (polynomial-length) “clue” or “guess” were appended to the problem input string. An important subclass of NP problems are referred to as *NP-complete* or Non-deterministic Polynomial time Complete problems [8]. The CPU time required to solve an NP-complete problem, based on known algorithms, grows exponentially with the “size” of the problem. There are no polynomial time transformations for NP-complete problems, nor are there any polynomial time algorithms capable of solving NP problems. The potential to solve NP and NP-complete problems depends on the availability of certain heuristics. One problem *polynomially reduced* to another if a polynomially bounded number of calls to an algorithm for the second will always solve the first. A problem $\Omega \in NP$ is shown to be NP-complete by polynomially reducing another already known NP-complete problem to Ω . Any problem Ω , whether a member of NP or not, to which we can polynomially transform an NP-complete problem is NP-hard.

Definition 5.28 (The CONTINUITY PROBLEM): Let Y be a synthesis mapping of a closure function space of $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ to a closure attribute space $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$. The decision problem, termed the *CONTINUITY PROBLEM*, concerns the existence of the following property: there exists a functional property f such that Y is continuous at f ; i.e., $F \subseteq \mathcal{F}, f \notin F, f \in U_{\mathcal{F}}(F)$ implies $Y(f) \in U_{\mathcal{D}}(Y(F))$. An analogous continuity problem can be defined for an analysis mapping.

Theorem 5.6: If the number of functional properties in \mathcal{F} is finite, the

CONTINUITY PROBLEM is NP-hard (in the cardinal number of the \mathcal{F}).

Following the proof of Theorem 5.6, we conclude that the Continuity Problem becomes undecidable for instances that have unbounded space of functional properties. The implication might be that, in general, a synthesis process can not be verified for continuity by an automatic problem solving procedure. Thus, the focus should be on developing computational methods that can be practically implemented for verifying the continuity of only a certain percentage of all possibilities (that is, the analysis and synthesis mappings be “almost” continuous).

Definition 5.29 (The HOMEOMORPHISM PROBLEM): The decision problem, termed the *HOMEOMORPHISM PROBLEM*, concerns the existence of a one-to-one analysis mapping Γ of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ and its inverse synthesis mapping $\Upsilon \equiv \Gamma^{-1}$ such that : $F \subseteq \mathcal{F}$ implies $\Upsilon(U_{\mathcal{F}}(F)) = U_{\mathcal{D}}(\Upsilon(F))$.

Theorem 5.7: If the number of functional properties in \mathcal{F} is finite, the HOMEOMORPHISM PROBLEM is NP-hard.

5.5 BASIS FOR FUNCTION AND ATTRIBUTE

5.5.1 DEFINITION AND PROPERTIES

For each functional property f , we define its closure as a set of functional descriptions in \mathcal{F} , each of which qualifies by logical inference the functional property f . We visualize a functional property $x \in U_{\mathcal{F}}(f)$ by thinking that x is “close” to f , in which case $U_{\mathcal{F}}(f)$ is called a “neighborhood” of f . In other words, given the overall design specification f , a number of alternative function decompositions may exist, each of which satisfies the overall design specification (similar interpretation is given to a closure attribute space).

A function space (or attribute space) can be very large and complicated. Often it simplifies matters to deal with a smaller collection of neighborhoods, which generates the entire “neighborhood system” (see Appendix A) of a function space (or attribute space) through unions. Such a collection is called a *basis*. The intuitive meaning of a basis for a closure function space (or attribute space) is that given an overall design specification f , the designer is able to precisely decompose it into corresponding functions (or attributes) through the unions in the basis.

The definition of a basis also has a heuristic value in design systems where the designer routinely encounters many design problems. In such cases, the designer can improve performance by extending the initial phase that involves specifying a basis for the corresponding function space. This strategy can serve to reduce the computational complexity of the search for the function space.

The definition of a basis follows:

Definition 5.30 (Basis): Let $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ be a closure function space. A *basis* for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ is a collection $\Phi = \{G_i\}_{i \in I}$ with the following property: for each set of functional properties F , there exists a sub-collection of Φ , $\{G_i\}_{i \in \Omega}$ such that $U_{\mathcal{F}}(F) = \bigcup_{i \in \Omega} U_{\mathcal{F}}(G_i)$. An analogous definition could be presented for an attribute space.

The whole collection of neighborhoods is also considered a basis. This fact is of little use since the point of defining a basis is to produce a *smaller* collection of neighborhoods with which to work.

Definition 5.31 (Function Space Character): For each basis Φ for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, $|\Phi|$ denotes the cardinality (number of members) of the basis Φ . The function space character, denoted by $Ca(\mathcal{F})$, is defined as the greatest lower bound of $|\Phi|$ for any basis Φ ; i.e., $Ca(\mathcal{F}) = \inf \{ |\Phi^i| : \Phi^i \text{ is a basis for } \langle \mathcal{F}, U_{\mathcal{F}} \rangle \}$.

Example 5.26: Consider the domain of fastener design. The designer would like get a better definition of the design specification *fastener coupling* (denoted by f). The designer decomposes the initial specification as follows: In order to design a shaft coupling, either *the nature of coupling is rigid \wedge the coupling is able to transmit a torque* (denoted by g_1), or *the nature of coupling is flexible \wedge the coupling is able to transmit a torque* (denoted by g_2). When this is the case, $U_{\mathcal{F}}(\{f\}) = U_{\mathcal{F}}(\{f, g_1\}) \cup U_{\mathcal{F}}(\{f, g_2\})$ and the sets $\{f, g_1\}$ & $\{f, g_2\}$ may be included in the basis of the related function space.

Next, we investigate the concepts of function space character and of basis. For brevity the results are stated in terms of function spaces. Analogous results can be obtained for attribute spaces.

Theorem 5.8: Let $G_1, G_2 \in \Phi$. If $f \in U_{\mathcal{F}}(G_1) \cap U_{\mathcal{F}}(G_2)$, then there exists $G \in \Phi$ such that $f \in U_{\mathcal{F}}(G) \subseteq U_{\mathcal{F}}(G_1) \cap U_{\mathcal{F}}(G_2)$.

Theorem 5.9: Assume $Ca(\mathcal{F}) \leq t$. Then for every collection $\{G_i\}_{i \in I}$, there exists a set $I_0 \subseteq I$ such that $|I_0| \leq t$ and $\bigcup_{i \in I_0} U_{\mathcal{F}}(G_i) = \bigcup_{i \in I} U_{\mathcal{F}}(G_i)$.

The next theorem shows that each basis for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ includes a “smaller” collection of subsets that forms a basis for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$.

Theorem 5.10: Let t be a cardinal number, and suppose $Ca(\mathcal{F}) \leq t$. Then, for

every basis Φ (for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$), there exists a basis Φ_0 such that $|\Phi_0| \leq t$ and $\Phi_0 \subseteq \Phi$.

Next, we show that the property “ Φ is a basis for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ ” is a topological invariant.

Theorem 5.11: Let the analysis mapping Γ be a one-to-one mapping of $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ onto $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, the synthesis mapping be the inverse of Γ (i.e., $\Upsilon \equiv \Gamma^{-1}$), and Γ be a *homeomorphism*. If $\Phi = \{G_i\}$ is a basis for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, then $\{\Upsilon(G_i)\}$ is a basis for $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$.

5.5.2 SPACE CHARACTER AS A DESCRIPTIVE COMPLEXITY MEASURE

In this section, we suggest that the function space character (see Definition 5.31) can gauge the descriptive complexity inherent in the space. The descriptive complexity of a given space measures the amount of information that is required to describe the related space. A descriptive complexity measure is dependent on the number of entities in the space and the variety of interdependence among the entities. Indeed, all else being equal, our ability to cope or comprehend with a system tends to decrease when the entities involved or their interconnections increase. There are many different ways in which descriptive complexity can be expressed. However, each of them has to satisfy the following properties:

Let \mathcal{C} be a collection of closure spaces. A descriptive complexity measure for \mathcal{C} is a single-valued relation W on \mathcal{C} with a range of \mathfrak{R} (where \mathfrak{R} denotes the set of real numbers) provided that the following conditions are satisfied:

1. $W(\phi) = 0$;
2. If U is a single-valued relation on 2^P with a range of 2^P , and Q is a subset of P ; then the single-valued relation $U|_Q$ on 2^Q with a range of 2^Q defined by $U|_Q(a) = U(a)$ is called the *restriction* of U to Q . If A is the closure space $\langle P, U \rangle$ and B is the closure space $\langle Q, U|_Q \rangle$, then $W(B) \leq W(A)$;
3. A is *homeomorphic* to B implies $W(A) = W(B)$.

Properties 1 and 2 ensure that descriptive complexity measures will be non-negative. Property 2 expresses the expectancy for a reduction in the complexity measure as a result of system abstraction. If the closure function space and the closure attribute space are homeomorphic, then it can be shown that the following property holds: for each $f \in \mathcal{F}$, F is a neighborhood of f if and only if $\Upsilon(F)$ is a neighborhood of $\Upsilon(f)$. Thus, the mutual dependence among the functional properties of \mathcal{F} is topologically

invariant. In view of this fact, Property 3 maintains that two topologically equivalent spaces are equally complex. By applying Definition 5.31 and Theorem 5.11, it can be shown that the function space character (see Definition 5.31) satisfies the properties stated above; and thus, it has the intuitive meaning of a descriptive complexity measure.

5.6 CONCLUDING REMARKS

In this chapter the intuitive concept of the design process as a mapping of the desired functionality of a product to the description of the final product without the intervention of a catalogue is formalized by introducing the notion of the idealized design process.

Regardless of whether or not the idealized design process serves as the basis for a descriptive theory of design, it also has a heuristic value in offering some guidelines for building CAD systems:

1. *Knowledge Representation* - there exist two potential artifact representations: extensional and intensional. In the extensional representation, a structural property (or functional property) is expressed as the set of artifacts having this property (e.g., as summarized in Table 5.1 and Table 5.2). In the intensional representation, artifacts are described by the set of attributes characterizing them. The idealized design process supports the use of intensional representation of artifacts via Propositional calculus which describes functional and structural properties. The ideal design process also suggests, through the notion of neighborhoods (that represent function or attribute decomposition groups), the use of hierarchical knowledge structures for design support systems (see Section 5.3.4). Graph structures are reminiscent of neighborhoods. Thus, employing such knowledge structures may result in better support of real design.

In the presence of a design catalogue, the synthesis process is made easier by the use of the catalogue as mediator between the specification and the design description. In this case, extensional representation may result in a better support of real design.

2. *Design Process* - The automation of design can be improved by providing a rule of direct correspondence between the function space and the attribute space without the intervention of a catalogue. Models and thorough research analysis (e.g., based on kinematics or mechanics) that are sufficiently detailed must be used for mediating between the attribute and its corresponding function.

The concepts presented in this chapter are related to subsequent chapters in several respects:

1. According to the continuity property explored in Section 5.4, when designers make an incremental change to the design, they expect that the resulting design will be consistent with the beginning design. After a design modification (redesign), the

constraint solver should honor the initial design choice. When the specifications are modified, we wish to not only find a new satisfactory design, we wish to find the *intended* design. This is what is meant by *consistent design*. The problem resides in selecting the correct solution. If there is only one possible solution to the specifications, then it is easy to maintain a consistent design. It is much harder if there are multiple competing solutions, all of which satisfy the specifications. Fortunately, the continuity property of design directs us towards a principle of design consistency: *small changes in specifications should lead to small changes in design*. Furthermore, large changes in specifications can often be decomposed to a series of small changes, in which case the principle can still be applied. In Part III of the book (Chapter 14), the concept of design consistency in the area of variational design is further formalized, and the COAST methodology is implemented for maintaining design consistency in those design areas where similarity between designs can be calculated;

2. The ideal design process does not address how the mapping between the function space and the attribute space happens in real design. In real design, processes progress from a set of *specifications* (describing the desired functions and constraints of the final product) and move towards the *artifact description* (the final detailed product description). This evolutionary nature of design is formalized in Chapter 6. In Chapter 6, we present the design process as a sequence of *synthesis steps*, where each step describes the “current” functional properties (specifications) and structural properties.

APPENDIX A - BASIC NOTIONS OF TOPOLOGY, AND LANGUAGE THEORY

This appendix defines the topological and Language Theory terms used in this chapter. The definitions listed were compiled from [5, 9, 10].

TOPOLOGICAL SPACES

closure operation If P is a set and U is a single-valued relation on 2^P with a range of 2^P , then we say that U is a closure operation for P provided that the following conditions are satisfied:

- (cl 1) $U(\emptyset) = \emptyset$;
- (cl 2) $X \subseteq U(X)$ for each $X \subseteq P$;
- (cl 3) $U(X \cup Y) = U(X) \cup U(Y)$

closure space A structure $\langle P, U \rangle$ where P is a set and U is a closure operation for P . If $\langle P, U \rangle$ is a closure space and $X \subseteq P$, then the set $U(X)$ is called the closure of X in $\langle P, U \rangle$.

closed set A subset X of a closure space $\langle P, U \rangle$ is called closed if $U(X) = X$.

open set A subset X of a closure space $\langle P, U \rangle$ is called open if its complement (relative to P) is closed; i.e., if $U(P - X) = P - X$. The collection \mathcal{O} of all open subsets of a closure space $\langle P, U \rangle$ fulfills the following three conditions:

- (o 1) The set P belongs to \mathcal{O} ;
- (o 2) The union of any sub-collection of \mathcal{O} belongs to \mathcal{O} ;
- (o 3) The intersection of any two members of \mathcal{O} belongs to \mathcal{O} .

interior operation If $\langle P, U \rangle$ is a closure space, then the interior operation int_U is a single-valued relation on 2^P with a range of 2^P such that for each $X \subseteq P$, $\text{int}_U(X) = P - U(P - X)$. The set $\text{int}_U(X)$ is called the interior of X in $\langle P, U \rangle$.

neighborhood A neighborhood of a subset X of a space $\langle P, U \rangle$ is any subset G of P containing X in its interior. The neighborhood system of a set $X \subseteq P$ in the space $\langle P, U \rangle$ is the collection of all neighborhoods of the set X .

limit point A limit point or an accumulation point of a set X in a closure space $\langle P, U \rangle$ is a point x belonging to the closure of $X - \{x\}$. By intuition a cluster point is a point where other points accumulate (or converge) to that point.

topological closure operation A closure operation U for P that satisfies the following condition: (cl 4) for each $X \subseteq P$, $U(U(X)) = U(X)$.

topological closure space A closure space $\langle P, U \rangle$ is said to be topological if the closure of P is topological. In a topological closure space $\langle P, U \rangle$ the closure of each set is closed in $\langle P, U \rangle$, and the interior of each subset of P is open in $\langle P, U \rangle$.

open basis An open basis of a topological space $\langle P, U \rangle$ is a collection \mathcal{B} of subsets of P such that a subset G of $\langle P, U \rangle$ is open if and only if it is the union of a sub-collection of \mathcal{B} .

convergence of a sequence A sequence $\{s_n\}$ in a topological space $\langle P, U \rangle$ converges to a point $s \in P$ if and only if for every neighborhood G of s there is a positive integer N_G such that $s_i \in G$ whenever $i > N_G$.

continuous mapping Let $\langle P, U \rangle$ and $\langle Q, V \rangle$ be two closure spaces. A mapping $f: P \rightarrow Q$ is continuous if for all $X \subseteq P$, $f(U(X)) \subseteq V(f(X))$.

homeomorphism If $\langle P, U \rangle$ and $\langle Q, V \rangle$ are two closure spaces, then a mapping $f: P \rightarrow Q$ is called a homeomorphism if and only if f is invertible and both f and f^{-1} are

continuous.

metric space A set S is called a metric space if with every pair of points $x, y \in S$, there exists a non-negative real number $d(x, y)$ that satisfies:

- (1) If $d(x, y) = 0$ then $x = y$ and $d(x, x) = 0$ always holds;
- (2) For any pair of points x, y ; $d(x, y) = d(y, x)$;
- (3) For any three points x, y ; and z , $d(x, z) \leq d(x, y) + d(y, z)$.

PROPOSITIONAL LOGIC

Logic is an indispensable part of reasoning or deductive thinking. Let us consider the *propositional logic*. A *proposition* is a declarative statement or sentence. A particular proposition may be either true or false. For example, consider the following two statements: "The word *blueberry* has two consecutive r's"; and "The word *peach* is six letters long." The former statement is true, and the latter statement is false.

The relationships between various declarative statements can be represented symbolically, using *connectors*. A connector is a function that makes a compound statement out of simple statements. The following connectors are defined as a matter of convention:

Symbol	Meaning	Connector name
1. \sim	"not"	Negation
2. \wedge	"and"	Conjunction
3. \vee	"or"	Disjunction
4. \Rightarrow	"if-then"	Implication
5. \Leftrightarrow	"if and only if"	Equivalence

Given propositions P and Q , the significance of the symbols are as follows:

1. $\sim P$ is true if P is false, or conversely false if P is true.
2. $P \wedge Q$ is true if both referent statements P and Q are true. If one or both P and Q is false, then the compound statement is false.
3. $P \vee Q$ is true if P or Q is true. The compound statement is false only when both P and Q are false.
4. $P \Rightarrow Q$ is false if P is true and Q is false, and true in all other cases. P is called the *antecedent* or *premise*; Q is called the *consequent* or *conclusion*.
5. $P \Leftrightarrow Q$ is true when P and Q are both true or both false; otherwise, the statement is false.

Based on the definitions given above; a *truth table*, which is a table of combinations of truth assignments to propositions, may be constructed. Table A.1 gives an example of such a truth table. Each column except those labeled by atomic formulas is derived from one or two columns to its left with the aid of the definitions

for $\sim, \wedge, \vee, \Rightarrow, \Leftrightarrow$. In this example, the third column is derived from the first, the fourth from the first and second, and the fifth from the third and fourth. This truth table shows that $(\sim P \Rightarrow (P \Rightarrow Q))$ is a tautology.

Table A.1 Example of A Truth Table

P	Q	$\sim P$	$(P \Rightarrow Q)$	$(\sim P \Rightarrow (P \Rightarrow Q))$
T	T	F	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

Since $P \Rightarrow Q$ and $\sim P \vee Q$ take on identical truth values for all the combinations of values for P and Q, they are equivalent; i.e., $(P \Rightarrow Q) \Leftrightarrow (\sim P \vee Q)$.

LANGUAGES

A “*symbol*” is an abstract entity that we shall not define formally; just as “point” and “line” are not defined in geometry. Letters and digits are examples of frequently used symbols. A *string* (or *word*) is a finite sequence of symbols juxtaposed. For example, $a, b,$ and c are symbols and $abcb$ is a string. The *length* of a string w , denoted $|w|$, is the number of symbols in the string. The empty string, denoted by ϵ , is the string consisting of zero symbols. Thus $|\epsilon| = 0$.

The *concatenation* of two strings is the string formed by writing the first string, followed by the second, with no separating space. For example, the concatenation of *white* and *house* is *whitehouse*. *Juxtaposition* is used as the concatenation operator. That is; if a and b are strings, then ab is the concatenation of these two strings. The empty string is the identity for the concatenation operator. That is, $\epsilon a = a\epsilon = a$ for each string a .

An *alphabet* is a finite set of symbols. A (*formal*) *language* is a set of strings of symbols from one alphabet. The empty set, \varnothing , and the set consisting of the empty string $\{\epsilon\}$ are languages. Note that they are distinct; the latter has a member while the former does not.

Let us define a regular expression and the language it represents. Let Σ be a finite alphabet. The *Regular expressions* over an alphabet Σ are the strings defined inductively as follows:

- (1) \varnothing is a regular expression representing the empty set of strings;
- (2) ϵ is a regular expression representing the set consisting of the empty string ϵ ;
- (3) For each $a \in \Sigma, a$ is a regular expression representing the set $\{a\}$;
- (4) If α and β are regular expressions representing sets A and B , then

- (i) $\alpha \cup \beta$ is a regular expression representing $A \cup B$;
(ii) $\alpha\beta$ is a regular expression representing the language $AB = \{xy: x \in A, y \in B\}$;
(iii) α^* is a regular language representing the language $A^* = \bigcup_{i=0}^{\infty} A^i$, where
 $A^0 = \{\varepsilon\}$ and $A^{i+1} = A^i A$ for $i \geq 0$.

APPENDIX B - BOUNDED POST CORRESPONDENCE PROBLEM (BPCP) [8]

INSTANCE: Finite alphabet Σ , two sequences of strings $a = (a_1, a_2, \dots, a_n)$ and $b = (b_1, b_2, \dots, b_n)$ from Σ^* , and a positive integer $K \leq n$.

QUESTION: Is there a sequence i_1, i_2, \dots, i_k where $k \leq K$ such that the two strings $a_{i_1} a_{i_2} \dots a_{i_k}$ and $b_{i_1} b_{i_2} \dots b_{i_k}$ are identical?

APPENDIX C - GRAPH ISOMORPHISM [8]

INSTANCE: Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

QUESTION: Is there a one-to-one mapping $f: V_1 \rightarrow V_2$, such that $(v_1, v_2) \in E_1$ if and only if $(f(v_1), f(v_2)) \in E_2$?

APPENDIX D - PROOFS OF THEOREMS

Proof of Theorem 5.1

Let us examine the set $B = \{F \in \mathcal{F}: F \subseteq U(F)\}$. F is not empty since $\emptyset \in F$. Define $F^* = \bigcup_{F \in B} F$. Let us show that $U(F^*) = F^*$.

Step 1: $F^* \subseteq \bigcup_{F \in B} U(F)$ -- $F \subseteq U(F)$ for every $F \in B$; hence, $F^* = \bigcup_{F \in B} F \subseteq \bigcup_{F \in B} U(F)$.

Step 2: $\bigcup_{F \in B} U(F) \subseteq U(F^*)$ -- For every $F \in B$, $F \subseteq F^*$. Hence, the monotonicity of U implies $U(F) \subseteq U(F^*)$ for every $F \in B$. Thus, it is concluded that $\bigcup_{F \in B} U(F) \subseteq U(F^*)$.

Step 3: $F^* \subseteq U(F^*)$ -- A direct result from Steps 1 and 2.

Step 4: $U(F^*) \subseteq F^*$ -- From step 3 we obtain $F^* \subseteq U(F^*)$ and the monotonicity of U implies $U(F^*) \subseteq U(U(F^*))$. Hence, $U(F^*) \in B$; i.e., $U(F^*) \subseteq F^*$.

Step 5: $U(F^*) = F^*$ -- A direct consequence of Steps 3 and 4. ■

Proof of Theorem 5.2

Since $F \subseteq \mathcal{F}$ implies $Y(U_{\mathcal{F}}(F)) = U_{\emptyset}(Y(F))$, we conclude that $Y(U_{\mathcal{F}}(F)) \subseteq U_{\emptyset}(Y(F))$. Thus the CS property is satisfied. The equivalence $Y(U_{\mathcal{F}}(F)) = U_{\emptyset}(Y(F))$ implies that $Y(U_{\mathcal{F}}(F)) \supseteq U_{\emptyset}(Y(F))$. Since Γ is a one-to-one mapping, we concluded that there exists a set for which $\Gamma(M) = F$. Therefore, we obtain $Y(U_{\mathcal{F}}(\Gamma(M))) \supseteq U_{\emptyset}(Y(\Gamma(M)))$. Since the synthesis mapping is the “inverse” of the analysis mapping (i.e., $Y(\Gamma(M)) = M$), we concluded that $Y(U_{\mathcal{F}}(\Gamma(M))) \supseteq U_{\emptyset}(M)$. Applying the analysis mapping to both sides, we obtain the CA property; i.e., $U_{\mathcal{F}}(\Gamma(M)) \supseteq \Gamma(U_{\emptyset}(M))$. ■

Proof of Theorem 5.3

Part 1: Let $F_1 \succ_{\mathcal{F}} F_2$. The OPS property implies $Y(F_1) \succ_{\emptyset} Y(F_2)$. Therefore, the OPAS property implies $\sim[U_{\emptyset}(Y(F_2)) \succ_{\emptyset} U_{\emptyset}(Y(F_1))]$. Since $Y(U_{\mathcal{F}}(F_2)) = U_{\emptyset}(Y(F_2))$ and $Y(U_{\mathcal{F}}(F_1)) = U_{\emptyset}(Y(F_1))$, by applying Theorem 5.2, we conclude $\sim[Y(U_{\mathcal{F}}(F_2)) \succ_{\emptyset} Y(U_{\mathcal{F}}(F_1))]$. Finally, by the OPS property, we obtain $\sim[U_{\mathcal{F}}(F_2) \succ_{\mathcal{F}} U_{\mathcal{F}}(F_1)]$, which concludes the proof. ■

Part 2: Let $M_1 \succ_{\emptyset} M_2$. The OPS property implies $\Gamma(M_1) \succ_{\mathcal{F}} \Gamma(M_2)$. Therefore, the OPFS property implies $\sim[U_{\mathcal{F}}(\Gamma(M_2)) \succ_{\mathcal{F}} U_{\mathcal{F}}(\Gamma(M_1))]$. Since $\Gamma(U_{\emptyset}(M_1)) = U_{\mathcal{F}}(\Gamma(M_1))$ and $\Gamma(U_{\emptyset}(M_2)) = U_{\mathcal{F}}(\Gamma(M_2))$, by applying Theorem 5.2, we conclude $\sim[\Gamma(U_{\emptyset}(M_2)) \succ_{\mathcal{F}} \Gamma(U_{\emptyset}(M_1))]$. Finally, by the OPS property, we obtain $\sim[U_{\emptyset}(M_2) \succ_{\emptyset} U_{\emptyset}(M_1)]$, which concludes the proof. ■

Proof of Theorem 5.4

Let $M_1 \subseteq M_2 \subseteq \mathcal{D}$. Since Y is a one-to-one mapping and additive, there exist sets F_1, F_2 such that $F_1 \subseteq F_2 \subseteq \mathcal{F}$ and $Y(F_1) = S_1$ & $Y(F_2) = S_2$. By Theorem 5.2, we obtain $Y(U_{\mathcal{F}}(F_1)) = U_{\emptyset}(Y(F_1))$ and $Y(U_{\mathcal{F}}(F_2)) = U_{\emptyset}(Y(F_2))$. The

monotonicity of $U_{\mathcal{F}}$ implies $U_{\mathcal{F}}(F_1) \subseteq U_{\mathcal{F}}(F_2)$. Since Υ is additive we obtain $\Upsilon(U_{\mathcal{F}}(F_1)) \subseteq \Upsilon(U_{\mathcal{F}}(F_2))$. Thus, $U_{\mathcal{D}}(\Upsilon(F_1)) \subseteq U_{\mathcal{D}}(\Upsilon(F_2))$; i.e., $U_{\mathcal{D}}(M_1) \subseteq U_{\mathcal{D}}(M_2)$ as required. ■

Proof of Theorem 5.5

Let F^* be a fixed-point of $U_{\mathcal{F}}$; i.e., $U_{\mathcal{F}}(F^*) = F^*$. By Theorem 5.2, $\Upsilon(U_{\mathcal{F}}(F^*)) = U_{\mathcal{D}}(\Upsilon(F^*))$ which also means $\Upsilon(F^*) = U_{\mathcal{D}}(\Upsilon(F^*))$. Thus, $\Upsilon(F^*)$ is a fixed-point of $U_{\mathcal{D}}$. ■

Proof of Theorem 5.6

It is easy to see that the CONTINUITY PROBLEM \in NP. Let us transform BOUNDED POST CORRESPONDENCE (BPC, see Appendix B) to the CONTINUITY PROBLEM. Let a finite alphabet Σ , sequences $a = \{a_1, a_2, \dots, a_n\}$ and $b = \{b_1, b_2, \dots, b_n\}$ of strings from Σ^* , and a positive integer $K \leq n$ form an arbitrary instance of BPC. We shall construct closure spaces $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ and $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, and a synthesis mapping Υ ; such that there exists a functional property f and that Υ is continuous at f if and only if the reply to BPC is the affirmative:

- Let N be the alphabet $N = \{1, 2, 3 \dots n\}$. The function and attribute spaces are defined as follows: $\mathcal{F} = \{a_1, a_2, \dots, a_n\}^K \cup \{1, 2, 3 \dots n\}^K$ and $\mathcal{D} = \{a_1, a_2, \dots, a_n\}^K \cup \{b_1, b_2, \dots, b_n\}^K \cup \{1, 2, 3 \dots n\}^K$; where $\{a_1, a_2, \dots, a_n\}^K$ denotes, for example, the set of all strings over $\{a_1, a_2, \dots, a_n\}$ of length less than or equal to K .

The closure operation $U_{\mathcal{F}}$ is defined as follows:

- $U_{\mathcal{F}}(a_{i_1} a_{i_2} \dots a_{i_k}) = (a_{i_1} a_{i_2} \dots a_{i_k})$; $U_{\mathcal{F}}(i_1 i_2 \dots i_k) = \{i_1 i_2 \dots i_k, a_{i_1} a_{i_2} \dots a_{i_k}\}$; and for each $F, G \subset \mathcal{F}$: $U_{\mathcal{F}}(F \cup G) = U_{\mathcal{F}}(F) \cup U_{\mathcal{F}}(G)$.

- The synthesis mapping, Υ , is defined as the identity mapping; i.e., for each set $F \subseteq \mathcal{F}$: $\Upsilon(F) = F$.

- The closure operation $U_{\mathcal{D}}$ is defined as follows:

$U_{\mathcal{D}}(i_1 i_2 \dots i_k) = \{i_1 i_2 \dots i_k, b_{i_1} b_{i_2} \dots b_{i_k}\}$; $U_{\mathcal{D}}(a_{i_1} a_{i_2} \dots a_{i_k}) = (a_{i_1} a_{i_2} \dots a_{i_k})$; and $U_{\mathcal{D}}(b_{i_1} b_{i_2} \dots b_{i_k}) = (b_{i_1} b_{i_2} \dots b_{i_k})$. Also, for each $M, N \subset \mathcal{D}$: $U_{\mathcal{D}}(M \cup N) =$

$$U_{\mathcal{D}}(M) \cup U_{\mathcal{D}}(N).$$

Now, it is easily verified that Υ has a single continuity point at $f = a_{i_1} a_{i_2} \dots a_{i_k}$ if and only if the reply to BPC is affirmative. ■

Comment: the BOUNDED POST CORRESPONDENCE PROBLEM is undecidable if no upper bound is placed on k ; e.g., see [10].

Proof of Theorem 5.7

Let us transform GRAPH ISOMORPHISM (see Appendix C) to the HOMEOMORPHISM PROBLEM. Let the graphs G_1 and G_2 form an arbitrary instance of GRAPH ISOMORPHISM. We construct closure spaces $\langle \mathcal{D}, U_{\mathcal{D}} \rangle$ and $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ such that there exists a one-to-one synthesis correspondence Υ if and only if the reply to GRAPH ISOMORPHISM is positive:

- The function and attribute spaces are defined as follows: $\mathcal{D} = V_1$ and $\mathcal{F} = V_2$.
- The closure operation $U_{\mathcal{F}}$ is defined as follows: $U_{\mathcal{F}}(\{v\}) = \{v\} \cup \{e_i : \langle v, e_i \rangle \in E_1\}$; and for each $F \subset \mathcal{F}$ and $G \subset \mathcal{F}$, $U_{\mathcal{F}}(F \cup G) = U_{\mathcal{F}}(F) \cup U_{\mathcal{F}}(G)$.
- The closure operation $U_{\mathcal{D}}$ is defined as follows: $U_{\mathcal{D}}(\{v\}) = \{v\} \cup \{e_i : \langle v, e_i \rangle \in E_2\}$; and for each $M \subset \mathcal{D}$ and $N \subset \mathcal{D}$: $U_{\mathcal{D}}(M \cup N) = U_{\mathcal{D}}(M) \cup U_{\mathcal{D}}(N)$.

Now, it is easily verified that there exists a homeomorphism Υ if and only if the reply to GRAPH ISOMORPHISM is the positive. ■

Proof of Theorem 5.8

Let $W = U_{\mathcal{F}}(G_1) \cap U_{\mathcal{F}}(G_2)$, then $f \in W = \bigcup_{i \in \Omega} U_{\mathcal{F}}(G_i)$ for some subset $\Omega \subseteq I$.

Now, $\exists i^* \in \Omega : f \in U_{\mathcal{F}}(G_{i^*})$; and since $\forall i \in \Omega : U_{\mathcal{F}}(G_i) \subseteq W$, we conclude $f \in U_{\mathcal{F}}(G_{i^*}) \subseteq U_{\mathcal{F}}(G_1) \cap U_{\mathcal{F}}(G_2)$. ■

Proof of Theorem 5.9

Take a generating set $\Phi = \{F_i\}$ for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ such that $|\Phi| \leq t$, and denote by Φ_0 the collection of all $F \in \Phi_0$ such that for some $i \in I$ we have $U_{\mathcal{F}}(F) \subseteq U_{\mathcal{F}}(G_i)$. To every $F \in \Phi_0$, define the set $\Pi(F) \subseteq I$ as $\Pi(F) = \{i: U_{\mathcal{F}}(F) \subseteq U_{\mathcal{F}}(G_i)\}$. By the axiom of choice, we can define a function $\pi(\cdot)$ such that to every $F \in \Phi_0: \pi(F) \in \Pi(F)$. Let us show that $I_0 = \pi(\Phi_0) \subseteq I$ satisfies the conditions of the lemma. First, observe that $|I_0| = |\pi(\Phi_0)| \leq |\Phi_0| \leq t$. Trivially, $\bigcup_{i \in I_0} U_{\mathcal{F}}(G_i) \subseteq$

$\bigcup_{i \in I} U_{\mathcal{F}}(G_i)$. Let us show the inverse inclusion. Take $f \in \bigcup_{i \in I} U_{\mathcal{F}}(G_i)$. Since Φ is a

generating set, $\exists I^* \in I, \exists F \in \Phi: m \in U_{\mathcal{F}}(F) \subseteq U_{\mathcal{F}}(G_i^*)$. That is, $F \in \Phi_0$ and $\pi(F) \in I_0$; i.e., $f \in U_{\mathcal{F}}(F) \subseteq U_{\mathcal{F}}(G_{\pi(F)}) \subseteq \bigcup_{i \in I_0} U_{\mathcal{F}}(G_i)$, which concludes the proof. ■

Proof of Theorem 5.10

Part 1: Suppose that $t \geq \omega_0$. Take a generating set $\Phi_1 = \{F_j\}_{j \in J}$ for the system $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ such that $|J| \geq t$. Let $\Phi = \{G_i\}_{i \in I}$, and for any $j \in J$ let $I(j) = \{i \in I: U_{\mathcal{F}}(G_i) \subseteq U_{\mathcal{F}}(F_j)\}$. Since Φ is a generating set for the system $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, we have $U_{\mathcal{F}}(F_j) = \bigcup \{U_{\mathcal{F}}(G_i): i \in I(j)\}$; and by Theorem 5.9, there exists a set $I_0(j) \subseteq I(j)$ such that $|I_0(j)| \leq t$ and $U_{\mathcal{F}}(F_j) = \bigcup \{U_{\mathcal{F}}(G_i): i \in I_0(j)\} = \bigcup \{U_{\mathcal{F}}(G_i): i \in I_0(j)\}$. Let $\Phi_0 = \{G_i: i \in I_0(j) \& j \in J\}$. Since $|J| \leq t$, from $|I_0(j)| \leq t$ and the equality $t^2 = t$, it follows that $|\Phi_0| \leq t$. Let us show that Φ_0 is a generating set for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$. Since Φ_1 is a generating set, we have -- for an arbitrary set $F \subseteq \mathcal{F}$ -- $U_{\mathcal{F}}(F) = \bigcup_{j \in \Omega} U_{\mathcal{F}}(F_j)$. Since $U_{\mathcal{F}}(F_j) = \bigcup \{U_{\mathcal{F}}(G_i): i \in I(j)\}$, we finally conclude $U_{\mathcal{F}}(F) = \bigcup_{j \in \Omega} \bigcup_{i \in I(j)} U_{\mathcal{F}}(G_i)$, which

proves that Φ_0 is a generating set for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$.

Part 2: Suppose $t < \omega_0$. Let $\Phi_1 = \{F_i\}_{i \in I}$ be a generating set for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ such that $|\Phi_1| = Ca(\mathcal{F}) = k \leq t$. For any $G_i \in \Phi_1$, there exists a subset $I_i \subseteq I$ such that $U_{\mathcal{F}}(F_i) = \bigcup_{j \in I_i} U_{\mathcal{F}}(G_j)$. Let us show that $\exists j_i \in I_i$ such that $U_{\mathcal{F}}(F_i) = U_{\mathcal{F}}(G_{j_i})$. If it is not true, then $\forall j \in I_i: U_{\mathcal{F}}(G_i) \subset U_{\mathcal{F}}(F_i)$. Since Φ_1 is a

generating set for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$, we conclude that there exists a proper subset $\Phi_0 \subset \Phi_1$ such that $U_{\mathcal{F}}(F_i) = \{\cup U_{\mathcal{F}}(F_j) : F_j \in \Phi_0 \& F_i \notin \Phi_0\}$. Thus, $\Phi_1 - \{F_i\}$ is a generating set for $\langle \mathcal{F}, U_{\mathcal{F}} \rangle$ and $Ca(\mathcal{F}) = k - 1$, in contradiction to the assumption that $Ca(\mathcal{F}) = k$. ■

Proof of Theorem 5.11

We have to show that for each subset M of \mathcal{Q} for some subfamily $\{L_i\}_{i \in \Omega}$ $U_{\mathcal{Q}}(M) = \bigcup_{i \in \Omega} U_{\mathcal{Q}}(\Upsilon(L_i))$. Since Υ is a one-to-one mapping $\exists F: M = \Upsilon(F)$. Applying Theorem 5.2, we obtain $\Upsilon(U_{\mathcal{F}}(F)) = U_{\mathcal{Q}}(\Upsilon(F))$. By definition, for some subfamily $\{G_i\}_{i \in \Omega}$, $U_{\mathcal{F}}(F) = \bigcup_{i \in \Omega} U_{\mathcal{F}}(G_i)$. By applying Υ to both sides we obtain, $\Upsilon(U_{\mathcal{F}}(F)) = \Upsilon(\bigcup_{i \in \Omega} U_{\mathcal{F}}(G_i)) = \bigcup_{i \in \Omega} \Upsilon(U_{\mathcal{F}}(G_i))$. Finally, since the closure function and attribute spaces are *homeomorphic*, we obtain $\bigcup_{i \in \Omega} \Upsilon(U_{\mathcal{F}}(G_i)) = \bigcup_{i \in \Omega} U_{\mathcal{Q}}(\Upsilon(G_i))$. Let us denote for each $i \in \Omega$, $L_i = \Upsilon(G_i)$ and conclude the proof. ■

REFERENCES

1. Yoshikawa, "General Design Theory and a CAD System," in T. Sata and E. Warman, editors, *Man-Machine Communication in CAD/CAM, Proceedings of the IFIP WG5.2-5.3 Working Conference*, Amsterdam, pp. 35-57, 1981.
2. Tomiyama, "From General Design Theory to Knowledge Intensive Engineering," *Artificial Engineering for Engineering Design, Analysis, and Manufacturing*, Vol. 8 (4), 1994.
3. Tomiyama and P.J.W. Ten Hagen, "Organization of Design Knowledge in an Intelligent CAD Environment", in J.S. Gero, editor, *Expert Systems in Computer-Aided Design*, Amsterdam, pp. 119-152, 1987.
4. Yoram Reich, "A Critical Review of General Design Theory," *Research in Engineering Design*, 1995.
5. Croom, F., *Principles of Topology*, Sounders College Publishing, Chicago, 1989.
6. Newell, A. and Simon, H. A., *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
7. Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA, 1980.
8. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1979.
9. Čech, E., *Topological Spaces*, Wiley, London, 1966.
10. Hopcroft, J. E., and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
11. Ulrich, K. T., *Computation and Pre-parametric Design*, Technical Report 1043, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, 1988.
12. Pugh S., *Total Design.*, Addison-Wesley, New York, 1990.
13. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.

CHAPTER 6

MODELING THE EVOLUTIONARY DESIGN PROCESS

The primary concept of FDT used in Chapter 5 is that the design process is a mapping of the desired set of *specifications* (requirements and constraints) onto the *artifact description*. This ideal view of the design process does not address how the mapping between the function space and the attribute space occurs in real design. Real design is an evolutionary process that progresses from a set of *specifications* toward the *artifact*. In this chapter, the evolutionary nature of design is formalized through a series of transformations (*process steps*) beginning with the leading specifications and resulting in a physical description of the artifact. There is precedence in the relationship between transformed states. Each state is driven by a set of production rules and produces a set of new specifications and/or a partial solution.

6.1 INTRODUCTION

Let us recall some of the very basic features of the design process as addressed in Part I of this book:

- Design begins with the recognition of needs, and the variation from an existing artifact state; along with the realization that some action must take place in order to bridge the gap.
- Designing an artifact can be considered a transition from concepts and ideas to concrete descriptions. In engineering design, such design descriptions can range from specifications in formal languages (computer-aided engineering systems, symbolic programming techniques associated with AI, hardware design/description languages), to quasi-formal notation (linguistic descriptions, qualitative influence graphs), to very informal and visual descriptions (functional block diagrams, flow-diagrams, engineering drawings).
- The designer is constantly faced with the problem of bounded rationality.
- The model of bounded rationality assumes limitations on the cognitive and information processing capability of the designer's decision making.
- The design specifications, which constitute the constraints of a design problem, may initially not be precise or complete. Hence, the evolution and elaboration of

these specifications becomes an integral part of the design process.

- Traditional engineering design methods generally use specifications of satisfactory performance (bounded rationality) rather than optimal performance (pure rationality).
- Constructive alternatives and design solutions are not provided in advance. Thus, they must be found, developed, and synthesized by some research process, which is iterative and evolutionary in nature.

As a consequence of the above features, we contend in Chapter 2 that the design process can be viewed as a stepwise, iterative, and evolutionary transformation process. The concepts underlying the evolutionary characteristic of design are captured in two main views: *ontogenetic* and *phylogenetic* design evolution. Ontogenetic design evolution refers to design processes that share observed evolutionary phenomenon. This condition occurs between the time specifications (requirements and constraints) are assigned to the designer and the time the artifact is passed on to the manufacturer. During this period, the artifact evolves and changes from the initial form to the acceptable form, and we say that there is a fit between the artifact and the specifications. Phylogenetic design evolution refers to redesign, which is defined as the act of successive changes or improvements made to a previously implemented artifact. An existing artifact is modified to meet the changes made to the original specifications.

In spite of the foregoing two views of design evolution and regardless of whether we are designing computer software, bridges, manufacturing systems or mechanical fasteners, their design adhere to the following evolutionary scheme. The design process begins with the establishment of *initial specifications* $\{ \theta_0 \}$. This first step is subjective but is clearly one of the most critical steps. Given the initial specifications, knowledge data, in the form of *known facts* and *production rules*, is provided by the designer as a means of achieving the specifications. This results in the inference of a partial design of the artifact that satisfies the specifications, and/or new sub-specifications and derived facts. The modification of a partial solution and/or specifications is performed in order to remove discrepancies and establish a fit between specifications and solution (see Figure 6.1). Eventually, a complete solution emerges, and the design is at the “full content” and specifications are at the “null state”. The evolved information reflects the fundamental feature of bounded rationality. The view that the hierarchical decomposition of specifications is performed along with the evolving hierarchies in the structural domain is consistent with many other design methodologies [e.g., 4-14].

The set of transformations that lead to the final, complete solution can be a useful guide to the development of general purpose, highly effective CAD tools. In addition, the set of transformations can constitute a knowledge-level process that explains the designer’s decision making process [see 11, 13]. This type of *sound explanatory* model is specified by the following [see also 16, 17]: Consider a design problem with initial conditions (specifically consisting of knowledge body K) and a resulting artifact; where the data about the design process is gathered by observation. Given the evolutionary model of the design process, an *explanation* of how the

artifact was generated will can be developed if a symbol transformation process (a path from the initial conditions to an accepting process state) can be described such that: (1) the symbol transformation process is in accordance with the model; (2) the symbol transformation process uses only the designer’s knowledge body; and (3) the symbol transformation process produces the artifact. By “sound,” we mean that the proposed model of the design process is an explanation for an entire domain of design problems of which the observed artifact is an instance or an element. The evolutionary model presented in this chapter is corroborated by the *case study* approach where a particular designing event is singled out for examination (e.g., see Chapter 3 and [13, 15]).

It is important to emphasize that the design process is *not a deterministic phenomenon*. In other words, if we place the designer of an artifact with exactly the same initial conditions and knowledge body, there is no guarantee that the same design process or artifact will evolve. The most important consequence of this is that a model of the design process cannot be tested by deriving predictions about what would happen under the model, and by empirically determining whether the predictions hold or not. This is in contrast with Popper’s (see Chapter 2) conclusion that the best reason for entertaining a theory is that it is testable (more accurately “falsifiable”); i.e., that it makes predictions that are *readily* refutable by evidence if the theory is false.

This chapter is organized as follows: Section 6.2 provides an overview of the model, and is followed by two illustrative examples. The formal model is presented in Section 6.3. The iterative (evolutionary) scheme of the design process is formalized, and a discussion of the correctness and computational complexity of the model is provided in Section 6.4. Section 6.5 concludes the chapter.

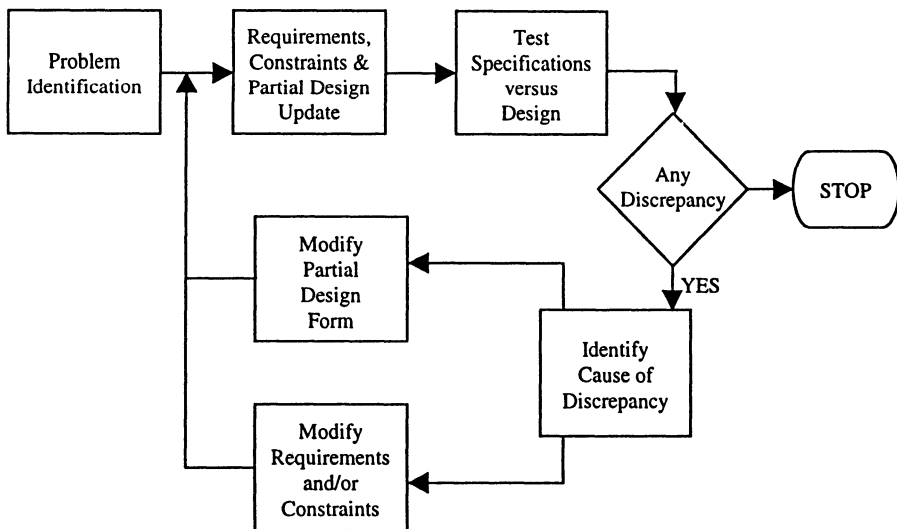


Figure 6.1 The Evolutionary Scheme of Generating New Specifications and Partial Design

6.2 PREVIEW OF THE MODELS

In this section, we construct three models for the design process. The detailed models and their analysis are provided in the next section (6.3). We first introduce a *type-1 design process*, which is defined as the mapping of the initial specifications of the product through a series of transformations that can be *either* decomposition of specifications *or* matching specifications with partial solutions. In a *type-0 design process* special restrictions are imposed on the form of transformations in a type-1 process. In a *type-2 design process* decomposition and synthesis transformations can be performed simultaneously and special restrictions are imposed on the form of transformations. Both type-0 and type-1 models are special cases of the type-2 model.

6.2.1 TYPE-1 DESIGN PROCESS

Definition 6.1 (Type-1 Design Process): We formally denote a type-1 design process as a *finite automaton* (See Appendix A) $DP \equiv \langle L, Q, P, T_A, T_S, S_0, F \rangle$, where

- L denotes the *design description*, which includes both structural and specification attributes, and is required for process state and production rule descriptions;
- Q is a finite set of *process states*;
- P is a finite set of *production rules*; which are retrieved from the designer's knowledge body, and are suggested as a means of changing the current process state to a modified state;
- The *analysis* transformation T_A is an operator that is used for decomposition of specifications. T_A is the *transition* function mapping $Q \times P$ to Q . That is, $T_A(S, p)$ is a process state for each process state S and rule p ;
- The *synthesis* transformation T_S is an operator that is used for matching the specifications with partial solutions. T_S is the transition function mapping $Q \times P$ to Q . That is, $T_S(S, p)$ is a process state for each process state S and rule p ;
- S_0 in Q is the *initial* process state;
- $F \subseteq Q$ is the set of *terminal* process states.

A process is a series of *transformations*. The series exhibits precedence among its transformed *process states*. A *process state* is described by $S \equiv \langle M, \theta \rangle$, where:

1. M denotes the *artifact description (artifact part)*, which is the tentative partial solution (set of structural properties) synthesized as yet in the process

(execution).

2. $\theta \equiv (\theta^P, \theta^V)$ denotes the *specification description (specification part)*. The specification part includes θ^P , which is the set of *presumed* specifications that remain to be satisfied; and θ^V , which is the set of *validated* specifications that are already satisfied.

At the beginning of the design process, the initial *specification part* exists in pure specification form and the artifact part has a null or empty content:

$$\text{Process State 0: } S_0 \equiv \langle \emptyset, (\theta_0^P, \theta_0^V) \rangle$$

A *process step (cycle)* corresponds to the *transformation (transition)* of one process state to another. A transformation, which describes the relation between two adjacent process states, is activated by a set of knowledge tokens in the form of *rules* included in the designer's *knowledge body*. Two transformation types are available:

- *Analysis* transformation T_A transforms the process state $\langle M_i, (\theta_i^P, \theta_i^V) \rangle$ to $\langle M_i, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ by applying the production rule p . That is $T_A: \langle M_i, (\theta_i^P, \theta_i^V) \rangle \times p \rightarrow \langle M_i, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ such that the new presumed specifications θ_{i+1}^P are consistent with θ_i^P .
- *Synthesis* transformation T_S transforms the process state $\langle M_i, (\theta_i^P, \theta_i^V) \rangle$ to $\langle M_{i+1}, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ by applying the production rule p . That is $T_S: \langle M_i, (\theta_i^P, \theta_i^V) \rangle \times p \rightarrow \langle M_{i+1}, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$.

As various transformations are performed, the specifications change and the design solution begins to take shape. The series of transformations ends with a *terminal* process state $S_n \in F$ in either of two cases:

1. The artifact part is transformed to a logically possible description (consistent description) that is also in its "full content," and the presumed specification part is transformed to the "null state." When this occurs, the terminal process state $S_n = \langle M_n, (\emptyset, \theta_n^V) \rangle$ is called *successful*, since all specifications have been implemented; otherwise,
2. If there are no possible modifications to either the artifact part or the specification part, then the process state $\langle M_n, (\theta_n^P, \theta_n^V) \rangle$ is called *failed*.

The *execution* of a design process is a series of transformations for process states $i=1,2,3,\dots,n$. For example, the following design process begins with the establishment of initial specifications θ_0 :

$$\text{Process State 0: } S_0 \equiv \langle \emptyset, (\theta_0^P, \theta_0^V) \rangle$$

$$\text{Process Step 1: } \langle \emptyset, (\theta_0^P, \theta_0^V) \rangle \xrightarrow{T_A, p_1} \langle \emptyset, (\theta_1^P, \theta_0^V) \rangle \equiv S_1$$

$$\text{Process Step 2: } \langle \emptyset, (\theta_1^P, \theta_0^V) \rangle \xrightarrow{T_A, p_2} \langle \emptyset, (\theta_2^P, \theta_0^V) \rangle \equiv S_2$$

$$\text{Process Step 3: } \langle \emptyset, (\theta_2^P, \theta_0^V) \rangle \xrightarrow{T_S, p_3} \langle M_1, (\theta_3^P, \theta_1^V) \rangle \equiv S_3$$

$$\text{Process Step 4: } \langle M_1, (\theta_3^P, \theta_1^V) \rangle \xrightarrow{T_S, p_4} \langle M_2, (\emptyset, \theta_1^V \wedge \theta_3^P) \rangle \equiv S_4$$

At the end of the 4th process step, the initial specifications θ_0 have been converted into the physical artifact M_2 , which contains all the blueprint information for the artifact's implementation.

The process *history* includes the sequence of all process states visited so far $\{S_0, S_1, S_2, \dots, S_n\}$, as well as the transformations and production rules used to modify the process states $\{\dots, (T_A, p_i), \dots, (T_S, p_k), \dots\}$.

Example 6.1 (Mechanical Fasteners Design): The design of mechanical fasteners is a common design problem in the realm of mechanical engineering. The function of a fastener is to hold two or more parts together by transmitting the load to the fastened parts. There are numerous types of existing mechanical fasteners (see Figure 6.2).

The proposed problem is to design a new fastener according to certain given specifications based on the knowledge that we already have from the existing fasteners. At the lowest level, a fastener might be described in terms of structural properties such as head radius, thread pitch, and thread depth (assuming that it has threads). Functional requirements, however, are usually not packaged so neatly. Instead, they are usually provided as high level requirements (such as the strength or precision of the fastener). From these high-level requirements, a more detailed description evolves through the design process.

1. *The Knowledge-Base:* The first step is to build a data base of knowledge. The existing designs are represented as a set of design descriptions, which include structural, functional, and causal relationships between structure and function. The structural description of the device and its components are listed. In the case of fasteners, the general hierarchy is composed of the five components that are common to all fasteners; drive, head, body, tail, and tip as shown in Figure 6.3.

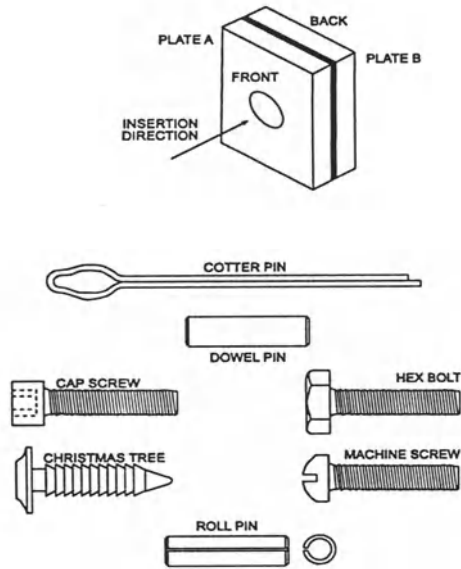


Figure 6.2 Seven Fasteners in the Knowledge Base (adapted from [4])

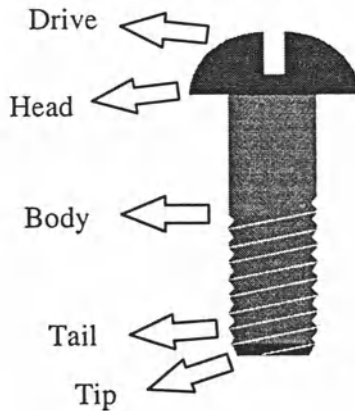


Figure 6.3 Hierarchy Structure of a Fastener

The behavioral abstraction of a design is usually determined by the structural hierarchy. For instance, the function of a fastener is to hold parts together, which is done by transmitting the load to the fastened parts. The relation is usually a causal one governed by physical laws, and the behavior of the components follows the same principle. Let us illustrate the relationships with the following example; the strength

of the fastener is determined by the static fatigue, stress rapture, and impact strength. The fatigue is determined by the material, head size, thread size, threaded length, fillet, fabrication, and surface structure. These properties, in turn, are determined by the physical attributes of the components and entity. A similar decomposition could be applied to the other properties of the fastener. This decomposition can be represented as the causal and factual knowledge of the system in a causal network and as an if-then structure (production rules). These rules link the functions to the structural attributes of the components.

2. *The Task:* To perform the synthesis task, the design process is used to transform functional specifications to a structural description. A typical input to the design process is a *conjunction* of functional attributes. The result of applying the above series of transformations to an input specification is a possible solution, which consists of a set of five structural attributes corresponding to the five parts of the fastener structure.

3. *A Run Time Design Process:* For our fastener design, the initial specification comes down as:

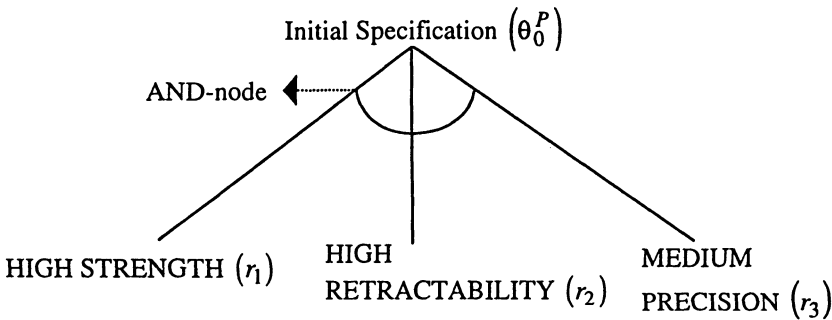


Figure 6.4 The Initial Specification

The first iteration (process step) may be to realize that strength is characterized separately by the head and tail of the fastener. By applying the knowledge-base of fasteners and analyzing the current requirements, we may conclude that both demand high strength:

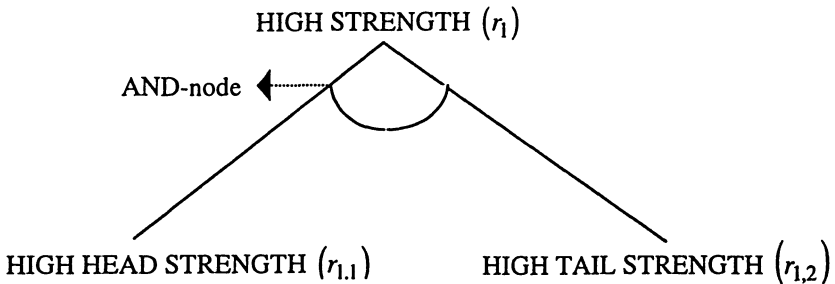


Figure 6.5 Decomposition of HIGH STRENGTH

The second iteration may be to improve the definition of retractability. There is a choice of how fasteners are driven and in order to get high retractability we need rotary-mode fasteners:

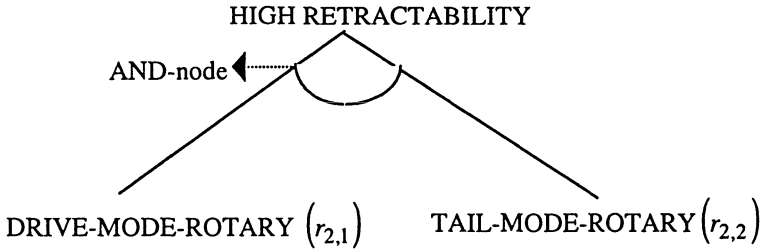


Figure 6.6 Decomposition of HIGH RETRACTABILITY

In the next iteration, we may realize that medium precision translates into a looser body fit:

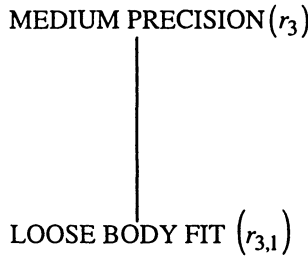


Figure 6.7 Decomposition of MEDIUM PRECISION

At some point, synthesis decisions begin to be made as the configuration of the artifact takes place. From the current information and the rules embedded in the knowledge base, we may infer in the next iteration that the head of the fastener needs a shoulder:

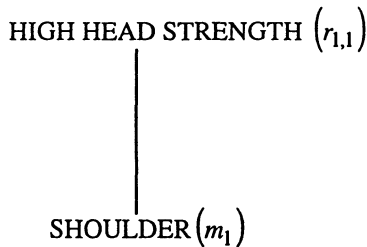


Figure 6.8 Matching the HIGH HEAD STRENGTH Requirement with the SHOULDER Structural Attribute

In the next iteration, we might infer that the tail of the fastener should be fastened

using threads and that the body should have threads:

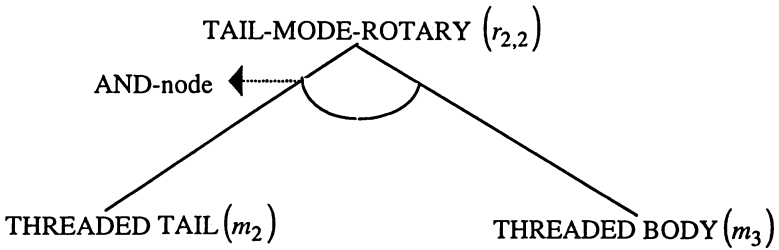


Figure 6.9 Matching the TAIL-MODE-ROTARY Requirement with the THREADED TAIL and THREADED BODY Structural Attributes

In the next iteration, we may infer that the drive type will be a hex head:

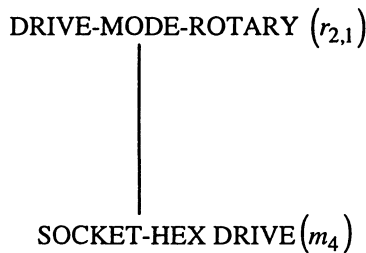


Figure 6.10 Matching the DRIVE-MODE-ROTARY Requirement with the SOCKET-HEX DRIVE Structural Attribute

In the next iteration, we find that the specification HIGH TAIL STRENGTH ($r_{1,2}$) and LOOSE BODY FIT ($r_{3,1}$) are *satisfied* by the structural attributes THREADED TAIL (m_2) and THREADED BODY (m_3). Since these structural attributes are already fixed, the artifact part remains unchanged and the specification part is transformed to the “null state.” Finally, by testing the tentative artifact part, we can infer that it does not contain all the information needed to function as a unit. A discrepancy still remains since the tip part has not been determined yet. The transition strategy is to determine the attribute for the tip, which is not in conflict with the already satisfied specifications. Hence, we may decide that the tip should be chamfered (m_5) in order to maximize reversibility. The process can similarly continue until all the physical attributes of the final fastener are determined.

The process *history* includes the sequence of all process states visited, and the transformations used to modify the process states. The process history of the fastener design is described in Table 6.1.

Table 6.1 The Process History of the Fastener Design

PROCESS STEP	ARTIFACT PART	PRESUMED SPECIFICATIONS	VALIDATED SPECIFICATIONS	TRANSITION
1	\emptyset	$r_1 \wedge r_2 \wedge r_3$	\emptyset	Analysis
2	\emptyset	$r_{1,1} \wedge r_{1,2} \wedge r_2 \wedge r_3$	\emptyset	Analysis
3	\emptyset	$r_{1,1} \wedge r_{1,2} \wedge r_{2,1} \wedge r_{2,2} \wedge r_3$	\emptyset	Analysis
4	\emptyset	$r_{1,1} \wedge r_{1,2} \wedge r_{2,1} \wedge r_{2,2} \wedge r_{3,1}$	\emptyset	Synthesis
5	m_1	$r_{1,2} \wedge r_{2,1} \wedge r_{2,2} \wedge r_{3,1}$	$r_{1,1}$	Synthesis
6	$m_1 \wedge m_2 \wedge m_3$	$r_{1,2} \wedge r_{2,1} \wedge r_{3,1}$	$r_{1,1} \wedge r_{2,2}$	Synthesis
7	$m_1 \wedge m_2 \wedge m_3$ m_4	$r_{1,2} \wedge r_{3,1}$	$r_{1,1} \wedge r_{2,1} \wedge r_{2,2}$	Synthesis
8	$m_1 \wedge m_2 \wedge m_3$ m_4	\emptyset	$r_{1,1} \wedge r_{1,2} \wedge r_{2,1} \wedge r_{2,2} \wedge r_{3,1}$	Synthesis
9	$m_1 \wedge m_2 \wedge m_3$ $m_4 \wedge m_5$	\emptyset	$r_{1,1} \wedge r_{1,2} \wedge r_{2,1} \wedge r_{2,2} \wedge r_{3,1}$	STOP

6.2.2 TYPE-2 DESIGN PROCESS

A type-2 design process permits simultaneous synthesis and analysis transformations. In this case, the distinction between “structural properties” and “specifications” becomes blurred and meaningless. The modifications needed to include simultaneous transitions are described as follows:

Definition 6.2 (Type-2 Design Process): We formally denote a type-2 design process as a *finite automaton* $DP = \langle L, Q, P, T, S_0, F \rangle$, where the design description L , the finite set of process states Q , the finite set of production rules P , the initial process state S_0 , and the set of terminal states F have the same meaning as in a type-1 design process. T is an operator, which is simultaneously used for decomposing specifications and matching specifications with partial solutions.

A process is a series of *transformations*. The series follows a precedence

relationship among its transformed *process states*. A *process state* is described by $S = \langle M_i \wedge \theta^P, \theta^V \rangle$, where $M_i \wedge \theta^P$ is a conjunction of structural attributes and presumed specifications, and θ^V represents validated specifications.

At the beginning of the design process, the initial *process state* exists in pure presumed specifications, and the artifact part and validated specifications have a null or empty content:

$$\text{Process State 0: } S_0 \equiv \langle \theta_0^P, \emptyset \rangle$$

A *process step (cycle)* corresponds to a *transformation (transition)* from one process state to another process state by applying a production rule p , i.e., $T : \langle M_i \wedge \theta_i^P, \theta_i^V \rangle \times p \rightarrow \langle M_{i+1} \wedge \theta_{i+1}^P, \theta_{i+1}^V \rangle$.

The series of transformations ends with a *terminal* process state S_n in either of two cases:

1. The artifact part is transformed to a logically possible (consistent) description that is also in its “full content,” and the presumed specification part is transformed to the “null state.” In this case, the terminal process state $S_n \equiv \langle M_n, \theta_n^V \rangle$ is called *successful*, since all specifications have been implemented;
2. There are no possible modifications to either the artifact or the specifications. In this case, the process state $\langle M_n \wedge \theta_n^P, \theta_n^V \rangle$ is called *failed*, since some specifications remain unsatisfied.

Example 6.2: Assume that the designer’s *knowledge body* is in the form of *rules* as in Table 6.2. Suppose that the initial specification is r_1 , and that an artifact must include the structural attribute m_1 . The designer first tries to find a rule that decomposes r_1 , i.e., a rule that has r_1 on its right-hand side (as a conclusion). The only candidate rules are 6 and 8, and 6 is chosen. At this new process step the designer establishes a new sub-goal of decomposing r_2 . If we can match r_2 with structural attributes, then r_1 would be satisfied by *modus ponens*. The next sub-goals are to decompose r_3 and m_1 ($r_3 \wedge m_1 \Rightarrow r_2$). m_1 is a structural attribute; thus, only r_3 needs to be decomposed. The entire process history can be seen in Table 6.3.

Table 6.2 Production Rules (P) for Example 6.2

Rule 1:	$m_2 \wedge r_4 \wedge r_5$	\Rightarrow	r_3
Rule 2:	$r_3 \wedge m_1$	\Rightarrow	r_2
Rule 3:	$m_3 \wedge r_7$	\Rightarrow	r_8
Rule 4:	m_1	\Rightarrow	r_6
Rule 5:	r_6	\Rightarrow	r_5
Rule 6:	r_2	\Rightarrow	r_1
Rule 7:	r_9	\Rightarrow	r_{10}
Rule 8:	$m_2 \wedge m_3$	\Rightarrow	r_1
Rule 9:	m_3	\Rightarrow	r_4

Table 6.3 Illustration of the Process History for Example 6.2

PROCESS STEP	$M_i \wedge \theta^P$	θ^V	RULE
1	r_1	\emptyset	6
2	r_2	\emptyset	2
3	$r_3 \wedge m_1$	\emptyset	1
4	$m_1 \wedge m_2 \wedge r_4 \wedge r_5$	\emptyset	9
5	$m_1 \wedge m_2 \wedge m_3 \wedge r_5$	r_4	5
6	$m_1 \wedge m_2 \wedge m_3 \wedge r_6$	r_4	4
7	$m_1 \wedge m_2 \wedge m_3$	$r_4 \wedge r_6$	STOP

Note that the designer could decompose r_1 in step one by applying rule 8 (since m_2 and m_3 are structural attributes). However, this process would terminate with the *failed* process state $\langle m_2 \wedge m_3, r_1 \rangle$, since the artifact solution does not include the structural attribute m_1 (i.e., the artifact part is not transformed to a “full content”). In this case, the system backtracks and rule 6 is tried.

6.3 DETAILED MODELING

In this section we elaborate on the models presented in the foregoing sections. This section will serve as a vehicle for implementing the model on a computer. It will also enable us to prove meta-theorems (i.e., theorems that tell us about properties of the design process) related to the *correctness*, *decidability* and the *complexity* of design processes (see Section 6.4).

6.3.1 TYPE-0 DESIGN PROCESS $\langle L, Q, P, T_A, T_S, S_0, F \rangle$

Design Description (L)

L is a propositional calculus (see [1] for an extensive presentation). The basic *vocabulary* and *syntax* of the *design description* consists of the following:

1. A finite set of names for structural and functional attributes (*unary predicates*). We choose to use the finite collection $m, m_1, m_2, \dots, m_{1,1}, \dots$ as the names for concrete structural attributes, and use the finite collection $r, r_1, r_2, \dots, r_{1,1}, \dots$ as the names for concrete functional attributes. These basic proposition letters are intended to be interpreted as the names for concrete structural attributes such as *the fastener will be fastened using threads*, or *the car will have hydraulic disk brakes* or *fuel injection*; or for concrete specifications such as *the drive mode is rotary*, or *the car has low maintenance costs*.
2. A name, \diamond , for a distinguished proposition that is known to be false.
3. The connectives \wedge, \vee, \neg , and \Rightarrow that are intended to be read as “conjunction,” “disjunction,” “negation,” and “implies,” respectively. These connectives can be used to combine simple structural and functional attribute letters, as in $(m_1 \wedge r_2) \Rightarrow (r_1 \wedge m_3)$.
4. We use the following conventions: (a) M_1, M_2, \dots stand for arbitrary expressions (artifact descriptions) that include only structural attributes; (b) $\theta_1, \theta_2, \dots$ stand for arbitrary expressions (specification descriptions) that include only functional attributes; and (c) $\alpha, \beta, \gamma, \alpha_1, \alpha_2, \dots$ denote expressions that may include structural and functional attributes.

The *semantics* of L are given in terms of two objects called T and F. We intend these to correspond to the intuitive notions of truth and falsity. The semantics of the basic letters and \diamond are given by providing a function V , called a *valuation*, whose domain is the set of letters plus \diamond and whose range is the set $\{T, F\}$. There are clearly an infinite number of such functions. The only constraint is that $V(\diamond)$ must be F. V defines the meaning of the basic expressions; i.e., the proposition letters and the symbol \diamond . The meaning of compound expressions can be constructed from the meanings of the basic ones that appear within them. For example, if $V(\alpha) = T$ and $V(\beta) = F$ then $V(\alpha \Rightarrow \beta) = F$.

L only has one inference rule; namely, that for any expressions α and β , you can infer β from α and $\alpha \Rightarrow \beta$. In addition to this inference rule, which is known as *modus ponens* or MP, there are *axioms* or expressions that are accepted as being true under any valuation. In other words, they are *valid*. There are axiom set choices that are equivalent.

The inferential component of L is completed by defining the notion of a proof. A *proof* of a conclusion β from hypotheses $\alpha_1, \alpha_2, \dots, \alpha_k$ is a sequence of expressions $\gamma_1, \gamma_2, \dots, \gamma_n$ with the following conditions:

1. β is γ_n ;
2. Each γ_i is either an *axiom*, or an α_j , or a result of applying the inference rule MP to γ_k and γ_l where k and l are less than i .

In such a situation we write, $\alpha_1, \alpha_2, \dots, \alpha_k \vdash \beta$. The following proof example should clarify the definition:

Example 6.3: Recalling Example 6.1, let us present a proof of $r_{1,1} \wedge r_{2,2}$ from $m_1 \wedge m_2 \wedge m_3$ and the rules given in Table 6.2:

1. Hypotheses: $m_1 \wedge m_2 \wedge m_3$ and the rules given in Table 6.2
2. m_2 (from #1 by an axiom)
3. m_3 (from #1 by an axiom)
4. $m_2 \wedge m_3$ (from #1 by an axiom)
5. $r_{2,2}$ (from #4 and rule $m_2 \wedge m_3 \Rightarrow r_{2,2}$ by MP)
6. m_1 (from #1 and an axiom)
7. $r_{1,1}$ (from #6 and rule $m_1 \Rightarrow r_{1,1}$ by MP)
8. $r_{1,1} \wedge r_{2,2}$ (from #5 and #7 and an axiom)

It is reasonable to have a proof of a conclusion β from an empty set of hypotheses. If this is the case, we generally say that we have a proof of β , or $\vdash \beta$. A formula that can be proved from an empty set of premises is often called a *theorem*. The completeness of L means that if β is valid then $\vdash \beta$. The following meta-theorem (the *Deduction Theorem*) is critical for deriving non-trivial proofs in L : if $\alpha_1, \alpha_2, \dots, \alpha_k \vdash \beta$ then $\alpha_1, \alpha_2, \dots, \alpha_{k-1} \vdash \alpha_k \Rightarrow \beta$.

A set of expressions $\alpha_1, \alpha_2, \dots, \alpha_k$ is *consistent* if $\alpha_1, \alpha_2, \dots, \alpha_k \vdash \diamond$ is not the case; otherwise, it is *inconsistent*. If an expression β is accepted as being false under any valuation (a *contradiction*) then $\beta \vdash \diamond$. ($\beta \vdash \diamond \equiv \vdash \sim \beta$).

Process State (Q)

Q is a finite set of *process states*. A *process state* $S \in Q$ is described by $S \equiv \langle M, \theta \rangle$. The *artifact part* M is an expression in L of the form $m_1 \wedge m_2 \wedge \dots \wedge m_k$. The *specification part* $\theta \equiv (\theta^P, \theta^V)$ consists of the set of *presumed* specifications

θ^P in L - of the form $\theta_1^P \wedge \theta_2^P \wedge \dots \wedge \theta_n^P$, and the set of *validated* specifications θ^V in L - of the form $\theta_1^V \wedge \theta_2^V \wedge \dots \wedge \theta_n^V$.

Production Rules (P)

The finite set of *production rules* P are expressions in L , which are suggested as a means of transforming a current process state to a modified state. The *production rules* P are retrieved from the designer's knowledge body. The production rules are either of the form $m_1 \wedge m_2 \wedge \dots \wedge m_k \Rightarrow r$ or $r_1 \wedge r_1 \wedge \dots \wedge r_1 \Rightarrow r$.

Analysis Transformation (T_A)

The *analysis transformation* T_A transforms a process state $\langle M_i, (\theta_i^P, \theta_i^V) \rangle$ to $\langle M_{i+1}, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ by adhering to the following rule: if θ_i^P is of the form $\theta_{i1}^P \wedge r \wedge \theta_{i3}^P$, and $\theta \Rightarrow r$ is a production rule of P , then transform θ_i^P to $\theta_{i+1}^P = \theta_{i1}^P \wedge \theta \wedge \theta_{i3}^P$ and θ_i^V to $\theta_{i+1}^V = \theta_i^V \wedge r$.

Synthesis Transformation (T_S)

The *synthesis transformation* T_S transforms a process state $\langle M_i, (\theta_i^P, \theta_i^V) \rangle$ to $\langle M_{i+1}, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ by adhering to the following rules:

1. Let θ_i^P be of the form $\theta_{i1}^P \wedge r \wedge \theta_{i3}^P$. If $m_1 \wedge m_2 \wedge \dots \wedge m_k \Rightarrow r$ is a production rule of P , then transform θ_i^P to $\theta_{i+1}^P = \theta_{i1}^P \wedge \theta_{i3}^P$, θ_i^V to $\theta_{i+1}^V = \theta_i^V \wedge r$, and M_i to $M_i \wedge m_1 \wedge \dots \wedge m_k$;
2. If $\theta_i^P = \emptyset$ and M_i does not contain all the information needed to function as a unit, then transform M_i to $M_i \wedge m$ (for a structural property m).

Execution and Terminal States

An *execution* is a series of process states beginning with an initial process state S_0 ;

i.e., $S_0, S_1, S_2, \dots, S_n$, such that either of the following is true:

1. $T_A(S_i, p) = S_{i+1}$;
2. $T_S(S_i, p) = S_{i+1}$.

The *terminal* (or *final*) process state $S_n \equiv \langle M_n, (\theta_n^P, \theta_n^V) \rangle$ satisfies one of the following conditions:

1. $\theta_n^P = \emptyset$; there is no proof of $\sim M_n$ and $\sim \theta_n^V$ (i.e., M_n and θ_n^V are consistent), and M_n contains all the information "needed to function as a unit." In this case S_n is called *successful*, since all specifications have been implemented;
2. $\theta_i^P \neq \emptyset$ and no analysis or synthesis transitions pertain to S_n . In this case S_n is called *failed*, since some specifications remain unsatisfied;
3. $\theta_n^P = \emptyset$ and there is proof of $\sim M_n$ or $\sim \theta_n^V$ (i.e., M_n or θ_n^V are inconsistent). In this case S_n is called *failed*, since the artifact cannot be implemented.

6.3.2 TYPE-1 DESIGN PROCESS $\langle L, Q, P, T_A, T_S, S_0, F \rangle$

In a type-1 design process no restrictions are imposed on the description of synthesis states, production rules, synthesis transformations, or terminal states. The only established restriction is related to the analysis transformation. The *analysis transformation* T_A transforms a process state $\langle M_i, (\theta_i^P, \theta_i^V) \rangle$ to $\langle M_i, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$ by adhering to the following rules:

1. $(\theta_{i+1}^P \wedge \theta_{i+1}^V) \vdash (\theta_i^P \wedge \theta_i^V)$; that is, the new specifications derive the current specifications;
2. if $M_i \vdash \theta_i^P$, then transform θ_i^P to $\theta_{i+1}^P = \emptyset$ and θ_i^V to $\theta_{i+1}^V = \theta_i^V \wedge \theta_i^P$.

6.3.3 TYPE-2 DESIGN PROCESS $\langle L, Q, P, T, S_0, F \rangle$

The design description L has the same meaning as for the type-0 process; but the finite set of process states Q , finite set of rules P , transformation T , and set of terminal process states F have to be redefined.

Process State (Q)

A process state $S \in Q$ is described as $S = \langle \alpha, \theta^V \rangle$, where the *design form* α is of the form $M \wedge \theta^P$. The artifact description M , the presumed specifications θ^P , and the *validated* specifications θ^V are all in conjunctive form.

Production Rules (P)

The designer's knowledge body includes the following type of *production* rules: $m_1 \wedge \dots \wedge m_k \wedge \theta_1 \wedge \dots \wedge \theta_l \Rightarrow r$.

Transformation (T)

T is the *transition* function mapping $Q \times P$ to Q . That is, $T(S, p)$ is a process state for each process state S and rule p . The *transformation* T transforms a process state $\langle \alpha_i, \theta_i^V \rangle$ to $\langle \alpha_{i+1}, \theta_{i+1}^V \rangle$ by adhering to the following rules:

1. Let $\alpha_i = M_i \wedge \theta_i^P$, where θ_i^P is of the form $\theta_{i1}^P \wedge r \wedge \theta_{i3}^P$. If $M \wedge \theta \Rightarrow r$ is a production rule of P , then transform α_i to $\alpha_{i+1} = M_i \wedge M \wedge \theta_{i1}^P \wedge \theta \wedge \theta_{i3}^P$;
2. If $M \Rightarrow r$ is a production rule of P , then transform α_i to $\alpha_{i+1} = M_i \wedge M \wedge \theta_{i1}^P \wedge \theta_{i3}^P$, and θ_i^V to $\theta_{i+1}^V = \theta_i^V \wedge r$;
3. If $\theta_i^P = \emptyset$ and M_i does not contain all the information needed to function as a unit, then transform M_i to $M_i \wedge m$ (for a structural property m).

Execution and Terminal States

An *execution* is a series of process states beginning with an initial process state $S_0 \equiv \langle \emptyset, \theta^V \rangle$; i.e., $S_0, S_1, S_2, \dots, S_n$, such that $T(S_i, p) = S_{i+1}$. The *terminal* (or *final*) process state $S_n \equiv \langle M_n \wedge \theta_n^P, \theta_n^V \rangle$ satisfies one of the following conditions:

1. $\theta_n^P = \emptyset$; there is no proof of $\sim M_n$ and $\sim \theta_n^V$ (i.e., M_n and θ_n^V are

consistent) and M_n contains all the information “needed to function as a unit.” In this case S_n is called *successful*, since all specifications have been implemented;

2. $\theta_i^P \neq \emptyset$ and the transition T does not pertain to S_n . In this case S_n is called *failed*, since some specifications remain unsatisfied;
3. $\theta_n^P = \emptyset$ and there is proof of $\sim M_n$ or $\sim \theta_n^V$ (i.e., M_n or θ_n^V are inconsistent). In this case S_n is called *failed*, since the artifact cannot be implemented.

6.4 CORRECTNESS AND COMPLEXITY OF THE DESIGN PROCESS

There are two fundamental issues of interest to researchers in computer assisted engineering design related to the inferential component of the design process. We would like to be reassured that a solution (artifact part) will satisfy the initial presumed specifications, and would also like to know how many “actions” (transformations) must be taken before reaching a solution. These issues are related to the *correctness* and *computational complexity* of design processes.

6.4.1 CORRECTNESS OF THE DESIGN PROCESS

In the following, the correctness of the foregoing design processes is established by constructing a proof of the initial presumed specifications (θ_0^P) from the terminal artifact solution (M_n).

Theorem 6.1: Correctness of type-1 and type-0 design processes

If an execution of a type-1 or type-0 design process ends with a successful process state S_n , then $M_n \vdash \theta_0^P$.

Proof: Let $S_0, S_1, S_2, \dots, S_n$ be an execution that begins with an initial process state S_0 . The proof follows directly from the following properties that hold for type-1 and type-0 processes:

Property 1. For every $i \in \{1, 2, \dots, n-1\}$: $(\theta_{i+1}^P \wedge \theta_{i+1}^V) \vdash (\theta_i^P \wedge \theta_i^V)$;

Property 2. $M_n \vdash \theta_n^V$.

Applying these properties, the following proof of θ_0^P from the terminal artifact

solution M_n can be constructed:

1. Hypotheses: M_n and the foregoing Properties 1 and 2
2. θ_n^V (from #1 and Property 2 by *modus ponens*)
3. $\theta_{n-1}^P \wedge \theta_{n-1}^V$ (from #2, Property 1, and $\theta_n^P = \emptyset$ by *modus ponens*)
- .
- .
- $n + 1$. $\theta_1^P \wedge \theta_1^V$ (from # n and Property 1 by *modus ponens*)
- $n + 2$. θ_0^P (from $n + 1$ by *modus ponens*, Property 1 and $\theta_0^V = \emptyset$)

Hence, $M_n \vdash \theta_0^P$, as required. ■

Theorem 6.2: Correctness of type-2 design processes

If an execution of a type-2 design process ends with a successful process state S_n then $M_n \vdash \theta_0^P$.

Proof: Let $S_0, S_1, S_2, \dots, S_n$ be an execution that begins with an initial process state S_0 . The proof follows directly from the following properties that hold for type-2 processes:

- Property 1. For $i \in \{1, 2, \dots, n-1\} : (M_{i+1} \wedge \theta_{i+1}^P \wedge \theta_{i+1}^V) \vdash (M_i \wedge \theta_i^P \wedge \theta_i^V)$;
 Property 2. $M_n \vdash \theta_n^V$.

Applying these properties, the following proof of θ_0^P from the terminal artifact solution M_n can be constructed:

1. Hypotheses: M_n and the foregoing Properties 1 and 2
2. θ_n^V (from #1 and Property 2 by *modus ponens*)
3. $M_n \wedge \theta_n^V$ (from #1 and #2)
4. $M_{n-1} \wedge \theta_{n-1}^P \wedge \theta_{n-1}^V$ (from #3, Property 1, and $\theta_n^P = \emptyset$ by *modus ponens*)
- .
- .
- $n + 2$. $M_{n-1} \wedge \theta_1^P \wedge \theta_1^V$ (from # $n + 1$ and Property 1 by *modus ponens*)

$$n + 3. \quad \theta_0^P \qquad \text{(from \# } n + 2, \text{ Property 1, } \theta_0^V = \emptyset \text{ and } M_0 = \emptyset \text{ by } \textit{modus ponens})$$

Hence, $M_n \vdash \theta_0^P$, as required. ■

6.4.2 COMPUTATIONAL COMPLEXITY OF THE DESIGN PROCESS PROBLEM

The process models presented in Section 6.3 imply that there is a set of discrete *process states* that can be described by a finite collection of *unary predicates* (structural and functional attributes). The application of a *production rule*, which is retrieved from the designer’s knowledge body, may (or may not) change the current process state; but a production rule always provides the same process state from the same prior process state. A particular process state is designated as the *initial* or *starting* state. Other states, possibly including the initial state, are designated as *terminal* or *accepting* states. Thus, a *design process (DP)* problem instance is determined by a finite state automaton whose initial process state, number of states, and set of accepting states are known to the problem solver. The task is to halt after producing an accepting path of transformations in the automaton from the initial process state to an accepting process state. To determine the accepting states, the problem solver uses effective tests.

In the analysis of computation theory [2], a *problem* is a collection of *instances* that share a mathematical form but differ in size. Whenever problems (such as the *DP* problem) can be represented by finite state automata whose nodes are state descriptions and whose arcs label the production rules (operators) that change the current process state, it is natural to measure the *size* of a problem instance by the total number of predicates (required for a process state description) and production rules of the problem instance.

When the automaton assumption is made, the number of predicates, n , required to describe a state determines the number of automaton states, 2^n . We are guaranteed that if there is an accepting path; there is one containing no more than $2^n - 1$ transformations (and production rules).

There are procedures (algorithms) that have the capacity to solve a particular instance of the *DP* problem, as well as all instances of the a problem. Since there are generally many different ways to solve a *DP* problem instance (e.g., by applying different production rules to each process state), the question of evaluation naturally arises. Are some ways of solving the *DP* problem better than others? In computation theory a common way to approach such questions is in terms of a complexity measure imposed on the procedure. One natural measure of a procedure’s complexity for solving a *DP* problem instance is by the number of effective tests needed to determine the accepting process states. How do the computational requirements of the procedure (i.e., number of tests) change when an intuitive or useful measure of the size of a problem instance is implemented? Does the *DP* problem have

alternative solution procedures that are more efficient than the procedure in question? Is there an “easy” solution to the *DP* problem?

Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources (such as computation time) required to solve problems with algorithms. Given a *DP* problem instance and a sequence of transformations (or production rules), the solution is assigned a zero value if there is a path from the initial process state to an accepting process state, and a one value otherwise. We convert the *DP* optimization problem of finding a sequence of transformations of minimum value to a *DP decision problem* by asking whether there is a feasible solution to the problem that has an objective function value that is equal to a specified threshold of zero or less.

The notion of “easy to verify” but “not necessarily easy to solve” decision problems is at the heart of the Class *NP* [2]. Specifically, *NP* includes all those decision problems that could be solved in polynomial-time if the right (polynomial-length) “clue” or “guess” were appended to the problem input string. One problem is polynomial time reducible to another if a polynomially bounded number of calls to an algorithm for the second will always solve the first. A problem Ω is *NP-hard* if all problems in *NP* are reducible by polynomial time to Ω . If Ω is *NP-hard* and $\Omega \in \text{NP}$, then we say Ω is *NP-complete* or Non-deterministic Polynomial time Complete problems [2]. The CPU time required to solve *NP-complete* and *NP-hard* problems, based on known algorithms, grows exponentially with the “size” of the problem. At this point in time, no polynomial time algorithms capable of solving *NP-complete* or *NP-hard* problems, and it is unlikely that polynomial time algorithms will be developed for these problems. The potential to solve *NP-complete* or *NP-hard* problems depends on the availability of certain heuristics.

A problem Ω is shown to be *NP-hard* by polynomially reducing another already known *NP-complete* problem to Ω . Thus, we can show that the *DP* decision problem is *NP-hard*. This can be done with the *3-CNF* problem; which is *NP-complete* [2], and polynomially reduces to a special case of the *DP* decision problem. The *3-CNF* decision problem is defined in the following subsection.

The 3-CNF Decision Problem [2]

A *Boolean expression* is an expression composed of variables, parentheses, and the operators \wedge (logical AND), \vee (logical OR) and \sim (negation). The precedence of these operators is \sim highest, then \wedge , then \vee . Variables take on values 0 (false) and 1 (true); as do expressions. If E_1 and E_2 are Boolean expressions, then the value of $E_1 \wedge E_2$ is 1 if both E_1 and E_2 have the value 1, and 0 otherwise. The value of $E_1 \vee E_2$ is 1 if either E_1 or E_2 has the value 1, and 0 otherwise. The value of $\sim E_1$ is 1 if E_1 is 0 and 0 if E_1 is 1.

A Boolean expression is said to be in *conjunctive normal form (CNF)* if it is of the form $E_1 \wedge E_2 \wedge \dots \wedge E_K$; and each E_i (called a *clause*) is of the form $t_{i1} \wedge t_{i2} \wedge \dots \wedge t_{in}$, where each t_{ij} is *literal* (either x or $\sim x$ for some variable x).

We usually write \bar{x} instead of $\sim x$. For example, $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4) \wedge \bar{x}_3$ is in CNF. The expression is said to be in 3-CNF if each clause has exactly three distinct literals.

An expression in 3-CNF is *satisfiable* if there is an assignment of 0's and 1's to the variables that gives the expression the value 1. Given an expression in 3-CNF, the 3-CNF problem is to determine whether the expression is satisfiable.

The Design Process Decision Problem is NP-HARD

The main result in this section concerns the computational complexity of the *design process* problem. Since a type-0 design process is a special case of both type-1 and type-2 processes, it is sufficient to prove that the *DP* decision problem is NP-hard when the type-0 process assumption is made. The precise *DP* decision computational problem is:

INSTANCE: A type-0 design process $\langle L, Q, P, T_A, T_S, S_0, F \rangle$ as defined in Section 6.3.1.

QUESTION: Is there a finite sequence of transformations and production rules; e.g., $\{\dots, (T_A, p_i), \dots, (T_S, p_n)\}$ that begins with the initial process state S_0 and ends with a *successful* terminal process state $S_n \in F$?

We measure the size of the *DP* decision problem instance by the total number of unary predicates in L and production rules in P . It is shown that:

Theorem 6.3: The *DP* decision problem is NP-hard.

Proof: We show that there is a polynomial transformation from the 3-CNF problem (as defined above) to an instance of the *DP* decision problem such that there is a Boolean-valued truth assignment to the 3-CNF problem if and only if there is an accepting path of transformations to the particular *DP* decision problem instance.

Let $\{x_1, x_2, \dots, x_N\}$ be a set of Boolean variables and $E = E_1 \wedge E_2 \wedge \dots \wedge E_K$ be a conjunction of clauses (each clause has exactly three distinct literals), which form an arbitrary instance of the 3CNF problem. Each E_i (called a *clause*) is of the form $t_{i1} \vee t_{i2} \vee t_{i3}$, and each t_{ij} is *literal*; that is, either x or $\sim x$ for some variable x . The construction of an instance of the *DP* decision problem goes as follows (an example follows the proof):

- Design Description (L) - Each clause $E_i = t_{i1} \vee t_{i2} \vee t_{i3}$ in the 3CNF instance corresponds to a structural attribute m_i in L . Thus, there are k structural attributes of this kind.

The functional attributes in L are in $Var \cup True \cup False$, where

$Var = \{X_1, X_2, \dots, X_N\}$; $True = \{T_1, T_2, \dots, T_N\}$; $False = \{F_1, F_2, \dots, F_N\}$

and the T 's ("true") or F 's ("false") are used to assign values to the "variables" X 's.

- Production Rules (P) - The production rules are of the form:

1. For every $j \in \{1, 2, \dots, N\}$: $T_j \Rightarrow X_j$;
2. For every $j \in \{1, 2, \dots, N\}$: $F_j \Rightarrow X_j$;
3. For every $j \in \{1, 2, \dots, N\}$: let $T_j \in True$ and $E_i = t_{i1} \vee t_{i2} \vee t_{i3}$ be the clause in the 3CNF instance that corresponds to the structural attribute m_i . We say that m_i satisfies T_j if the literal x_j appears in E_i . Define the set of all the predicates that satisfy T_j as $\{m_{j1}, m_{j2}, \dots, m_{jk}\}$, and let $T_j \Rightarrow m_{j1} \wedge m_{j2} \wedge \dots \wedge m_{jk}$ be a rule in P ;
4. For every $j \in \{1, 2, \dots, N\}$: let $F_j \in True$. We say that m_i satisfies F_j if the literal $\sim x_j$ appears in E_i . Define the set of all the predicates that satisfy F_j as $\{m_{j1}, m_{j2}, \dots, m_{jk}\}$, and let $F_j \Rightarrow m_{j1} \wedge m_{j2} \wedge \dots \wedge m_{jk}$ be a rule in P ;

- Initial Process State (S_0) - Let the initial process state be $\langle M_i, (\theta_0^P, \theta_0^V) \rangle$, where $M_i = \emptyset$, $\theta_0^P = X_1 \wedge X_2 \wedge \dots \wedge X_N$, and $\theta_0^V = \emptyset$;
- Analysis Transformation (T_A) - Let $S_i = \langle M_i, (\theta_i^P, \theta_i^V) \rangle$ such that θ_i^P is of the form $\theta_{i1}^P \wedge X_j \wedge \theta_{i3}^P$, and let $p \in P$ be the rule $\theta \Rightarrow X_j$ (θ is either T_j or F_j). Then, the process state S_i is transformed to $T_A(S_i, p) = \langle M_i, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$, where $\theta_{i+1}^P = \theta_{i1}^P \wedge \theta \wedge \theta_{i3}^P$ and $\theta_{i+1}^V = \theta_i^V \wedge X_j$;
- Synthesis Transformation (T_S) - Let $S_i = \langle M_i, (\theta_i^P, \theta_i^V) \rangle$ such that $\theta_i^P = \theta_{i1}^P \wedge \theta \wedge \theta_{i3}^P$ and $\theta \in True \cup False$. If $p \in P$ is the rule $\theta \Rightarrow m_{j1} \wedge m_{j2} \wedge \dots \wedge m_{jk}$, then the process state S_i is transformed to $T_S(S_i, p) = \langle M_{i+1}, (\theta_{i+1}^P, \theta_{i+1}^V) \rangle$, where $M_{i+1} = M_i \wedge m_{j1} \wedge m_{j2} \wedge \dots \wedge m_{jk}$, $\theta_{i+1}^P = \theta_{i1}^P \wedge \theta_{i3}^P$ and

$$\theta_{i+1}^V = \theta_i^V \wedge \theta;$$

- Terminal Process States (F) - The process state $S_n \equiv \langle M_n, (\theta_n^P, \theta_n^V) \rangle$ is terminal if $\theta_n^P = \emptyset$. S_n is successful if $M_n = m_1 \wedge m_2 \wedge \dots \wedge m_N$ (that is, M_n is in its “full content”) and failed otherwise.

It is easy to verify that the above construction can be done in polynomial time.

It remains to show that the DP automaton halts after producing an accepting path of transformations *if and only if* there is a Boolean-valued truth assignment of 0's and 1's to the variables $\{x_1, x_2, \dots, x_N\}$ that gives the expression E the value 1:

(IF) Let $\langle M_n, (\theta_n^P, \theta_n^V) \rangle$ be a successful process state. Since $\theta_0^P = X_1 \wedge X_2 \wedge \dots \wedge X_N$, $\theta_n^P = \emptyset$, and the analysis transition T_A uses rules of the form $T_j \Rightarrow X_j$ or $F_j \Rightarrow X_j$; it is concluded that θ_n^V is of the form $X_1 \wedge X_2 \wedge \dots \wedge X_N \wedge \theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_N$, where each θ_j is either T_j or F_j . We match “ X_j ” with the variable “ x_j ”, “ T_j ” with the value “1”, and “ F_j ” with the value “0”. It is easy to show that the Boolean-valued truth assignment to the variables x_j , which corresponds to the θ_j 's, gives the expression E the value 1. Let $\theta_j \Rightarrow m_{j1} \wedge m_{j2} \wedge \dots \wedge m_{jk_j} \quad \forall j \in \{1, 2, \dots, N\}$ be the rules in P used by the synthesis transformation T_S . By definition, the Boolean-valued truth assignment that corresponds to θ_j satisfies each clause that corresponds to some predicate in $\{m_{j1}, m_{j2}, \dots, m_{jk_j}\}$. Since $M_n = m_1 \wedge m_2 \wedge \dots \wedge m_N$, we must have $\bigcup_{j=1}^N \{m_{j1}, m_{j2}, \dots, m_{jk_j}\} = \bigcup_{j=1}^N m_j$. Thus, it is concluded that the Boolean-valued truth assignment corresponding to the θ_j 's satisfies each clause E_i and thus the expression $E = E_1 \wedge E_2 \wedge \dots \wedge E_K$. ■

(ONLY IF) Let $\{a_1, a_2, \dots, a_N\}$ be a Boolean-valued truth assignment (each a_j is either 1 or 0) for the variables $\{x_1, x_2, \dots, x_N\}$ that gives the expression $E = E_1 \wedge E_2 \wedge \dots \wedge E_K$ the value 1. Define the set of all the clauses that are satisfied by the Boolean-valued truth assignment a_j as $\{E_{j1}, E_{j2}, \dots, E_{jk_j}\}$. Let the predicates $\{\theta_1, \theta_2, \dots, \theta_N\}$ and $\{m_{j1}, m_{j2}, \dots, m_{jk_j}\}$ correspond to $\{a_1, a_2, \dots, a_N\}$ and $\{E_{j1}, E_{j2}, \dots, E_{jk_j}\}$, respectively. It is easy to show that the following path of rules

generates a successful process state:

- Process Step 1. $\theta_1 \Rightarrow X_1$;
 Process Step 2. $\theta_2 \Rightarrow X_2$;
 .
 .
 Process Step N $\theta_N \Rightarrow X_N$;
 Process Step $N + 1$ $\theta_1 \Rightarrow m_{11} \wedge m_{12} \wedge \dots \wedge m_{1k_1}$;
 .
 .
 Process Step $2N$ $\theta_N \Rightarrow m_{N1} \wedge m_{N2} \wedge \dots \wedge m_{Nk_N}$;

This concludes the proof of Theorem 6.3 ■

Example 6.4: Let $\{x_1, x_2, x_3, x_4\}$ be a set of Boolean variables and $E = (x_1 \vee x_2 \vee \sim x_3) \wedge (\sim x_1 \vee \sim x_3 \vee \sim x_4) \wedge (\sim x_2 \vee x_3 \vee x_4)$ be a conjunction of three clauses, which form an instance of the 3CNF problem. We show that a solution to this 3-CNF problem instance can be determined by solving the following instance of the DP decision problem:

- Design Description (L) - Each clause E_i in the 3CNF instance corresponds to a structural attribute m_i in L . Thus there are 3 structural attributes of this kind.

The functional attributes in L are in $Var \cup True \cup False$, where $Var = \{X_1, X_2, X_3, X_4\}$; $True = \{T_1, T_2, T_3, T_4\}$; $False = \{F_1, F_2, F_3, F_4\}$

- Production Rules (P) - The production rules are of the form:

- | | |
|---|-------------------------------------|
| 1. For every $j \in \{1, 2, 3, 4\} : T_j \Rightarrow X_j$ | 6. $T_4 \Rightarrow m_3$ |
| 2. For every $j \in \{1, 2, 3, 4\} : F_j \Rightarrow X_j$ | 7. $F_1 \Rightarrow m_2$ |
| 3. $T_1 \Rightarrow m_1$ | 8. $F_2 \Rightarrow m_3$ |
| 4. $T_2 \Rightarrow m_1$ | 9. $F_3 \Rightarrow m_1 \wedge m_2$ |
| 5. $T_3 \Rightarrow m_3$ | 10. $F_4 \Rightarrow m_2$ |

- The following path of rules and transformations generates a successful process state:

PROCESS STEP	ARTIFACT PART (M_i)	PRESUMED SPECIFICATIONS (θ_i^P)	VALIDATED SPECIFICATIONS (θ_i^V)	TRANSITION RULE (p)
1	\emptyset	$X_1 \wedge X_2 \wedge X_3 \wedge X_4$	\emptyset	$T_1 \Rightarrow X_1$
2	\emptyset	$T_1 \wedge X_2 \wedge X_3 \wedge X_4$	X_1	$F_2 \Rightarrow X_2$
3	\emptyset	$T_1 \wedge F_2 \wedge X_3 \wedge X_4$	$X_1 \wedge X_2$	$T_3 \Rightarrow X_3$
4	\emptyset	$T_1 \wedge F_2 \wedge T_3 \wedge X_4$	$X_1 \wedge X_2 \wedge X_3$	$F_4 \Rightarrow X_4$
5	\emptyset	$T_1 \wedge F_2 \wedge T_3 \wedge F_4$	$X_1 \wedge X_2 \wedge X_3 \wedge X_4$	$T_1 \Rightarrow m_1$
6	m_1	$F_2 \wedge T_3 \wedge F_4$	$X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge T_1$	$F_2 \Rightarrow m_3$
7	$m_1 \wedge m_3$	$T_3 \wedge F_4$	$X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge T_1 \wedge F_2$	$T_3 \Rightarrow m_3$
8	$m_1 \wedge m_3$	F_4	$X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge T_1 \wedge F_2 \wedge T_3$	$F_4 \Rightarrow m_2$
9	$m_1 \wedge m_2 \wedge m_3$	\emptyset	$X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge T_1 \wedge F_2 \wedge T_3 \wedge F_4$	STOP

As concluded in Theorem 6.3, the Boolean-valued truth assignment $x_1 = "1"$, $x_2 = "0"$, $x_3 = "1"$, and $x_4 = "0"$, which corresponds to the predicates in θ_n^V that are members of $True \cup False$ (i.e., $T_1, F_2, T_3,$ and F_4), satisfies the expression $E = (x_1 \vee x_2 \vee \sim x_3) \wedge (\sim x_1 \vee \sim x_3 \vee \sim x_4) \wedge (\sim x_2 \vee x_3 \vee x_4)$.

6.5 SUMMARY

In this chapter, the "real" nature of design, which is evolutionary, is formalized by a finite automaton representation. The finite automaton representation takes an instance of a problem where the arc labels signify production rules available to the design problem solver, nodes signify process states of the design resulting from transitions,

final process states represent accepting states, and initial process state represents the initial specifications introduced to the design process solver. A mechanism provides the problem solver with a description of a process state (required for specifications and partial solution description) that results from any sequence of transitions the problem solver undertakes. Each transition is driven by a set of rules and produces a new process state. The task is to halt after producing an accepting path of transformations in the automaton; that is, a path from the initial process state to an accepting process state. To determine the accepting states, the problem solver uses an effective test.

Theorem 6.1, 6.2 reassured that the formal model will not permit us to arrive at a solution (artifact part) that does not satisfy the initial presumed specifications (correctness). Theorem 6.3 teaches us that there is something inherently difficult about the DP problem in terms of how many “actions” (transformations) must be taken by the design process solver before it succeeds; even when the number of unary predicates (structural and functional attributes) and production rules is manageable. This theorem concludes that although expressiveness of the design description is necessary to enable the generation of a wide variety of solutions, simply increasing the expressiveness of a design description swamps the designer with alternative solution paths. Consequently, design problem solving requires a recourse to *heuristic* search procedures. In Chapter 10, we present an “intelligent” computerized advisory tool for problem decomposition, which is based on the framework developed here. In Chapter 20, we illustrate the application of the evolutionary design model to several nontrivial “real world” design problems.

APPENDIX A - BASIC NOTIONS OF AUTOMATA THEORY [ADAPTED FROM 3]

A *finite automata (FA)* consists of a finite set of states and transitions from state to state that occur on input symbols chosen from an alphabet Σ (see Appendix A in Chapter 5). For each input symbol there is exactly one transition out of each state (possibly back to the state itself). One state, usually denoted q_0 , is the initial state in which the automaton starts. Some states are designated as final or accepting states.

A directed graph, called a *transition diagram*, is associated with an FA as follows. The vertices of the graph correspond to the states of the FA. If there is a transition from state q to state p on input a , then there is an arc labeled a from state q to state p in the transition diagram. The FA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from the start state to an accepting state.

We formally denote a *finite automaton* by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, q_0 in Q is the *initial state*, $F \subseteq Q$ is the set of *final states*, and δ is the *transition function* mapping $Q \times \Sigma$ to Q . That is, $\delta(q, a)$ is a state for each state q and input symbol a .

REFERENCES

1. Ramsay, A., *Formal Methods in Artificial Intelligence*. Cambridge: Cambridge University Press, 1988.
2. Garey, M. R. and Johnson, D. S., *Computers and Intractability: A guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1979.
3. Hopcroft, J. E. and Ullman J. D., *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley Publishing Company, 1979.
4. Ulrich, K. T., *Computation and Pre-parametric Design*, Technical Report 1043, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, 1988.
5. Takeda, H., Veerkamp P., T. Tomiyama, and H. Yoshikawa, "Modeling Design Processes," *AI Magazine*, 11(4), pp. 37-48, 1990.
6. Umeda, Y., Takeda H., Tomiyama T., and H. Yoshikawa, "Function, Behavior, and Structure," in J. S. Gero (Ed.), *Applications of Artificial Intelligence in Engineering V*, Vol. 1. Berlin: Springer Verlag, pp. 177-194, 1990.
7. Takeda, H., Tomiyama T., and Yoshikawa H., "A Logical and Computable Framework for Reasoning in Design," in D. L. Taylor and L. A. Stauffer (Eds.), *Design Theory and Methodology - DTM'92*, ASME, pp. 167-174, 1992.
8. Pahl, G. and Beitz, W., *Engineering Design*. London: The Design Council, 1984.
9. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.
10. Chandrasekaran, B., "Design Problem Solving: A Task Analysis," *AI Magazine*, Winter, 1990.
11. Tomiyama, T., "From General Design Theory to Knowledge-Intensive Engineering," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1994.
12. Sriram, D. and Cheong, K., "Engineering Design Cycle: A Case Study and Implications for CAE," In *Knowledge Aided Design*. New York: Academic Press, 1990.
13. Takeda, H., Hamada, S., Tomiyama, T., and Yoshikawa, H., "A Cognitive Approach to the Analysis of Design Processes," In *Design Theory and Methodology - DTM'90* - , DE-Vol.27, (Rinderle, J. R., Ed.). New York: ASME, pp. 153-160, 1990.
14. Takeda, H., Tomiyama, T., and Yoshikawa, H., "Logical Formalization of Design Processes for Intelligent CAD Systems," In *Intelligent CAD. II*, (Yoshikawa, H., and Holden, T., Eds.), Amsterdam: North-Holland, pp. 325-336, 1990.
15. Dasgupta, S., "Testing the Hypothesis Law of Design: The Case of the Britannia Bridge," *Research in Engineering Design*, Vol. 6, pp. 38-57, 1994.
16. Thagard, P., *Computational Philosophy of Science*. Cambridge: MIT Press, 1988.
17. Thagard, P., "Explanatory Coherence," *Behavioral and Brain Sciences*, Vol. 12, pp. 435-467, 1989.

CHAPTER 7

GUIDED HEURISTICS IN ENGINEERING DESIGN

In Chapter 6, the desired function and constraints are mapped to the artifact description using an evolutionary process that can be visualized as a feedback loop of analysis, synthesis and evaluation. In this chapter, we define “*basic synthesis*” as the complete description of ‘primitive’ components and their relations so as to meet a set of specifications of satisfactory performance. To determine if “basic synthesis” could scale up to large problems, it is appropriate to analyze the computational complexity of the “basic synthesis” task - an issue which has often been ignored by the design research community. A special case of the “basic synthesis” activity, called the *Basic Synthesis Problem* (BSP), is addressed. The BSP is shown to be NP-Complete, generally applicable over all design domains, which suggests that the combinatorial complexity would be exponential, hence intractable within most modern computing environments. Such a theoretical mathematical analysis ignores critical domain-specific engineering knowledge. We show that by combining domain-specific mechanical engineering heuristics, which constrain the structure of potential artifacts, the BSP will be computationally tractable. In order to demonstrate the *guided heuristics* approach to specific domains, the heuristically guided combinatorial analysis will be presented for 2-D *wireframe feature recognition* systems which are predominant in industrial CAD systems.

7.1 INTRODUCTION

The previous chapters develop a generic formalism that aims to explain how design artifacts are represented, and how design processes conceptually perform in terms of knowledge manipulation (as captured by the ASE-based design paradigm). In Chapter 5, the term “design” is defined as a process. Given a description of a desired function and constraints, called specifications, provide a representation of an artifact that produces the function and satisfies the constraints. This representation, called artifact description or artifact structure, is identified as an algebraic structure. The mapping from functional requirements to artifact structure is identified in Chapter 6 as an evolutionary process consisting of similar activities (modeled through finite-state automata) repeated in a cyclic fashion. We address here a special case of the synthesis activity called the Basic Synthesis Problem (BSP). The term “basic synthesis” used in this study is defined as the complete specifications of ‘primitive’ components and their relations so as to meet a set of specifications of satisfactory

performance. To determine if any such “basic synthesis” could scale-up to large databases, it is appropriate to analyze the computational complexity of the basic synthesis problem - an issue which has often been ignored by the design research community. We show that the decision problem concerning the existence of a ‘satisfying’ artifact is NP-Complete. A naïve combinatorial analysis, generally applicable over all design domains, suggests that the combinatorial complexity would be exponential, hence intractable within most modern computing environments. Such a theoretical mathematical analysis ignores critical domain-specific engineering knowledge. It is argued that this theoretical potential for combinatorial explosion can be addressed by combining domain-specific engineering heuristics with a detailed combinatorial analysis. An example of such a heuristically *guided combinatorial* analysis will be presented by enforcing certain constraints on the artifact structure. This example presents combinatorial upper-bounds on the number of possible design solutions which indicate that BSPs upon large databases will be computationally tractable. These bounds are also instrumental for devising a heuristic strategy for the BSP. Thus our results are guidelines for developing algorithms to search for optimal and sub-optimal design solutions.

In order to demonstrate the applicability of the heuristically guided combinatorial analysis to a specific design problem, we present such an analysis for the domain of wireframe feature recognition, which “can be characterized as enhancing the geometric database representation of a mechanical artifact to include some design intent” [10]. A number of feature recognition systems have successfully demonstrated the ability to capture such design knowledge. This example presents combinatorial upper bounds which indicate that feature recognition upon large industrial part databases will be computationally tractable.

The rest of the chapter is organized as follows: Section 7.2 addresses the BSP and proves its computational intractability. The consequences are then explored. In Section 7.3, we examine the class of Constrained Basic Synthesis Problems (CBSP). We use an information-theoretic approach to derive a universal upper bound on the number of possible design solutions. In Section 7.4, we derive a refined upper bound on the number of possible design solutions, assuming a probabilistic search strategy for solving the CBSP. In Section 7.5, we use an heuristically guided combinatorial analysis to derive tractable computational upper bounds on feature recognition upon large databases. Section 7.6 concludes the chapter.

7.2 THE BASIC SYNTHESIS PROBLEM (BSP)

The rest of the chapter addresses and examines the computational complexity attached to a special case of the synthesis activity termed the *Basic Synthesis Problem* (BSP). Rigorous description of this problem will serve to illuminate the intractability properties of the design process.

7.2.1 PROBLEM FORMULATION

The term ‘basic synthesis’ is defined as the complete specifications of primitive components and their relations so as to meet a set of specifications of satisfactory performance, which correctly implies the domain-independent nature of the design as a generic activity. For example, a configuration can be described in terms of part types (a group of objects that are similar but have different sizes). Every part can be described by a set of attributes. Each attribute can be described by its dimension. The basic synthesis problem would be to find a collection of part types with their dimensions so as to meet a set of specifications.

Traditional engineering design methods prefer to use specifications of satisfactory performance over using optimal performance, since the designer is constantly faced with the problem of bounded rationality [3]. The model of bounded rationality takes as self-evident limitations on the cognitive and information processing capability of the designer’s decision making.

In practice, designers often set some criteria of satisfactoriness and if the design meets the criteria, the design problem is considered to have been ‘solved’. These criteria are mostly conflicting and heterogeneous in the same sense that the quality of the compared alternatives cannot be adequately expressed by a single integral criterion formed as a composition of the original (partial) criteria. Therefore, single-criterion optimization is often insufficient for choosing the best designs. The multiple criteria can include manufacturing and marketing considerations, in addition to traditional measures.

Example 7.1: Consider a multistorey reinforced concrete building design. A complete evaluation of a design solution (limited to the main beams) mainly involves the evaluation of the following four types of criteria [2]:

- Slenderness of beams. This checks the adherence of the design to zoning regulations regarding span/depth restrictions of the beams.
- Interference. This is based on the clearance between floor and roof, which in turn, is decided by the deepest beam in the plan.
- Beam-column compatibility. At every junction of the beam and column, a check is made to see if the smaller of the two columns dimensions is the same as the breadth of the beam that frames into that side. The evaluation may be based on the percentage of all such instances that violate this requirement.
- Adjacency - The desirability of two adjacent beams having the same depth is sought for. The evaluation may be based on the percentage of all such instances that violate this requirement.

The multiple criteria optimization is modeled through the evaluation operator. The evaluation operator measures the degree of efficiency of an artifact, defined as the degree of closeness (the less the better) to the actual specifications desired. Formally,

Definition 7.1 (Evaluation Operator): A mapping $E_{\theta} : M^* \rightarrow \mathfrak{R}^n$; θ is an

ordered set of specifications alternatives and $E_{\theta}(m) = (E_{\theta_1}(m), E_{\theta_2}(m), \dots, E_{\theta_n}(m))$, such that

$E_{\theta_i}(m)$ measures the degree of closeness of to the specification θ_i . E_{θ} induces a *preference structure* on the artifact space, i.e.,

1. $E_{\theta_i}(m_1) < E_{\theta_i}(m_2) \Leftrightarrow$ The designer strictly prefers module m_1 over module m_2 , relative to the specification attribute θ_i .
2. $E_{\theta_i}(m_1) = E_{\theta_i}(m_2) \Leftrightarrow$ The designer is indifferent regarding choice of m_1 over module m_2 , relative to the specification attribute θ_i .

The designer often defines, prior to solving the BSP, a *threshold* vector $\underline{K} = (k_1, k_2, \dots, k_n)$. \underline{K} represents (pointwise) the maximum degree of closeness (to θ), which is still accounted efficient. We term \underline{K} as the designer's *aspiration level*. Having defined the aspiration level, the BSP is formulated as the decision problem concerning the existence of a module evaluated *below* the aspiration level. Formally,

Definition 7.2 (Basic Synthesis Problem): Given a set of modules M , a set of relations C and a positive vector $\underline{K} \in \mathfrak{R}^n$, are there subsets $M_0 \subseteq M$, $C^0 \subseteq C$ and module $m \in \langle M_0, C^0 \rangle$ such that $E_{\theta}(m) \leq \underline{K}$?

Special instances of the BSP include PCB's design ("packing", "placement" and "routing"); 0 logic gates circuit satisfiability; and certain graph enumeration and isomorphism problems in the realm of mechanisms design. Towards examining thoroughly the computational aspects of the BSP, let us first identify one problem instance of the BSP.

Example 7.2 (Minimizing Microinstruction Size) [5, 6, 7]:

Microprogrammed Control

Microprogramming is a technique for implementing the control function of a processor in a systematic and flexible manner. Every instruction in a CPU is implemented by a sequence of one or more sets of concurrent microoperations. For example, a microoperation represented by the symbol $\text{Reg}_1 \leftarrow \text{Reg}_2$, when executed by the control unit, causes the content of the specified register Reg_2 to be gated to the register Reg_1 . Each microoperation is associated with a specific set of control lines which, when activated cause that microoperation to take place. Since the number of instructions and control lines is often in the hundreds, a *hardwired* control unit that select and sequences the control signals can be exceedingly complicated.

Microprogramming may be considered as an alternative to hardwired control

circuits. The control signals to be activated at any time are specified by a word called a *microinstruction* which is fetched from a *control memory* COM in much the same way an instruction is fetched from the main memory. A set of related microinstructions is called a *microprogram*.

Parallelism in Microinstructions

Microinstruction length is determined, at large, by the maximum number of simultaneous microoperations that must be specified. Therefore, microinstructions are often designed to take advantage of the fact that at the microprogramming level, many microoperations can be performed in parallel. For example, if microoperation m_1 writes into a register/store which is read by m_2 than m_1 and m_2 cannot be executed in parallel. Therefore, it is useful to divide the microoperation specification part of the microinstruction into k disjoint parts called *control fields*. Each control field encodes or represents a set of microoperations, any one of each can be executed concurrently with the microinstructions specified by the remaining control fields.

Minimally Encoded Microinstruction Organization (MEMO)

Let us examine the problem of encoding the control fields such that the total number of bits in the control fields is a minimum [5]. Let I_1, I_2, \dots, I_m be a set of microinstructions for the computer that is being designed. Each microinstruction specifies a subset of the available microoperations $S = \{S_1, S_2, \dots, S_n\}$ which must be activated. An encoded control field can activate only one microoperation at a time. Two microoperations S_1 and S_2 can be included in the same control field only if they cannot be executed in parallel from the same microinstruction. Call S_1 and S_2 a *compatible pair*. A *compatible class* H_1 is a set of microoperations that are pairwise compatible. Each H_1 can, then, be encoded in a single field of the microinstructions using $B_i = \lceil \log_2 |H_i| + 1 \rceil$ bits. The total length of the microinstruction would be $B = \sum_{i=1}^k B_i$ bits.

The problem is, to determine a set H_{\min} of compatible sets such that the corresponding length B , is the minimum. This problem is formulated in a BSP form as follows:

- The primitive modules (M) are identified with the set of microoperations S ;
- There are n types of relations (C), such that $c_j = \{ \langle S_{i_1}, S_{i_2}, \dots, S_{i_j} \rangle : \{ S_{i_1}, S_{i_2}, \dots, S_{i_j} \}$ is a compatible class that includes j microoperations }.
- A solution for the BSP is specified in terms of a set of relations $\{c_j\}$.

- The evaluation operator (E) is the total length (in bits) of the microinstruction that corresponds to the BSP solution.

Special Case

Let $S = \{s_1, s_2, \dots, s_8\}$ be the set of primitive modules. Consider the following pairs of microoperations that can be executed concurrently without any conflict in their resource usage:

$\langle s_1, s_2 \rangle, \langle s_1, s_3 \rangle, \langle s_2, s_4 \rangle, \langle s_2, s_5 \rangle, \langle s_2, s_6 \rangle, \langle s_2, s_7 \rangle, \langle s_2, s_8 \rangle, \langle s_3, s_5 \rangle, \langle s_3, s_6 \rangle, \langle s_4, s_1 \rangle, \langle s_4, s_6 \rangle, \langle s_5, s_7 \rangle, \langle s_6, s_7 \rangle, \langle s_7, s_1 \rangle, \langle s_7, s_2 \rangle$

BSP: Is there a module (a collection of compatible sets), such that the total length of the microinstruction would be ≤ 6 bits?

Two design alternatives are considered. The microinstruction that corresponds to the first alternative yields a total length of 7 bits:

$$c_1 = \{\langle s_1 \rangle\}; c_2 = \{\langle s_4, s_7 \rangle, \langle s_2, s_3 \rangle\}; c_3 = \{\langle s_5, s_6, s_8 \rangle\}$$

Alternative 2 yields a satisfying solution (with total microinstruction length of 6 bits):

$$c_2 = \{\langle s_2, s_3 \rangle\}; c_3 = \{\langle s_4, s_7, s_8 \rangle, \langle s_1, s_5, s_6 \rangle\}$$

7.2.2 THE INTRACTABILITY OF THE BSP

The technical result in this section concerns the computational complexity of the BSP. Over the past decade, complexity theory has emerged from a branch of computer science almost unknown to the operations research community into a topic of widespread interest and research. Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources - such as computation time, space and hardware size - required to solve problems via digital algorithms. The notion of “easy to verify” but “not necessarily easy to solve” decision problems is at the heart of the Class NP. Specifically, NP includes all those decision problems that could be polynomial-time solved if the right (polynomial-length) “clue” or “guess” were appended to the problem input string. An important subclass of NP problems are referred to as *NP-complete* or Non-deterministic Polynomial time Complete problems [1]. The CPU time required to solve an NP-complete problems, based on known algorithms, grows exponentially with the “size” of the problem. There exists no polynomial time transformations for NP-complete problems, nor are there any polynomial time algorithms capable of solving any NP problems. The potential to solve NP and NP-complete problems depends on the availability of certain heuristics.

In the following, the time complexity of solving the BSP is expressed as a

function of the size of the problem. By the size of the BSP we mean the total number of modules and relations ($|M| + |C|$). We assume that the input length for an instance of a BSP is efficiently encoded (i.e., the problem size grows polynomially with the number of modules and relations). We also assume that each formula $c_i(\langle m_{i1}, m_{i2}, \dots, m_{in} \rangle_k)$ and E_θ can be verified and computed respectively in polynomial time. It is shown that:

Theorem 7.1: The BSP is NP-complete.

Proof: Let us prove the first part of the theorem. It is easy to see that the BSP \in NP, since a nondeterministic algorithm need only guess subsets $M_0 \subseteq M, C^0 \subseteq C$ and a module $m \in \overline{\langle M_0, C^0 \rangle}$ and check in polynomial time that the threshold condition $E_\theta(m) \leq K$ is satisfied.

Next, we transform the *satisfiability problem* (see the appendix) to the BSP. Let $X = \{x_1, x_2, \dots, x_N\}$ be a set of Boolean variables and $E = \{e_1 \wedge e_2 \wedge \dots \wedge e_L\}$ be a conjunction of clauses (a clause is defined as the disjunction of literals over X), which form an arbitrary instance of the satisfiability problem. We construct sets M, C , an Evaluation mapping E_θ , and a positive integer K such that there are subsets $M_0 \subseteq M, C^0 \subseteq C$ and a module $m \in \overline{\langle M_0, C^0 \rangle}$ satisfying $E_\theta(m) \leq K$. The construction goes as follows (depicted in Figure 7.1):

- (1) Let $M = \{x_1, x_2, \dots, x_N\}$; (2) Let $C = \{\text{TRUE}, \text{FALSE}\}$, such that $\text{TRUE} = \{\langle x_{i_1}, x_{i_2}, \dots, x_{i_m} \rangle\}$ means $x_{i_j} = \text{“true”}$ (hence $\neg x_{i_j} = \text{“false”}$) for every $1 \leq j \leq m$ (“FALSE” is defined similarly); (3) a design solution is defined as $m = \{\text{TRUE} = \{\langle x_{i_1}, x_{i_2}, \dots, x_{i_m} \rangle\}, \text{FALSE} = \{\langle x_{j_1}, x_{j_2}, \dots, x_{j_{N-m}} \rangle\}\}$ such that $\{x_{i_1}, x_{i_2}, \dots, x_{i_m}\} \cup \{x_{j_1}, x_{j_2}, \dots, x_{j_{N-m}}\} = X$ and $\{x_{i_1}, x_{i_2}, \dots, x_{i_m}\} \cap \{x_{j_1}, x_{j_2}, \dots, x_{j_{N-m}}\} = \emptyset$; (4) The evaluation mapping is defined as

$$E_\theta(m) = \begin{cases} 0 & \text{if } E \text{ is satisfiable by } m \\ 1 & \text{otherwise.} \end{cases}$$

Now, one easily verifies that $\exists m: E_\theta(m) \leq 0 \Leftrightarrow E$ is satisfiable. ■

Theorem 7.1 above implies that, in the worst case, the time required to produce a solution is $O(k^n)$ where k is a constant and n , a parameter characterizing the ‘size’ of the problem. Therefore the use of heuristics (e.g., branch-and-bound techniques or backtrack search) to search for an optimal solution is inevitable when the problem is

large.

The intractability of the BSP infers that the number of potential designs is combinatorial -- that is, designs are collections of primitive elements, and many different elements can be combined in exponentially different ways:

Proposition 7.1 (Upper Bound): Let \mathfrak{S} denotes the set of possible solutions (a subset of $\overline{\langle M, C \rangle}$), $c_i \subseteq (M^*)^{n_i}$, $|M| = N$ and $|C| = \rho$. Then $|\mathfrak{S}| \leq \prod_{j=1}^{\rho} 2^{N^{n_j}}$.

Proof: Every relation c_i consists of assignments, each associated with the Cartesian product of n copies of the set M^* . Since $|(M^*)^{n_i}| = N^{n_i}$, the number of possible assignments is given by $2^{(N^{n_i})}$. As a design solution is the set of ρ relations, the required bound is concluded. ■

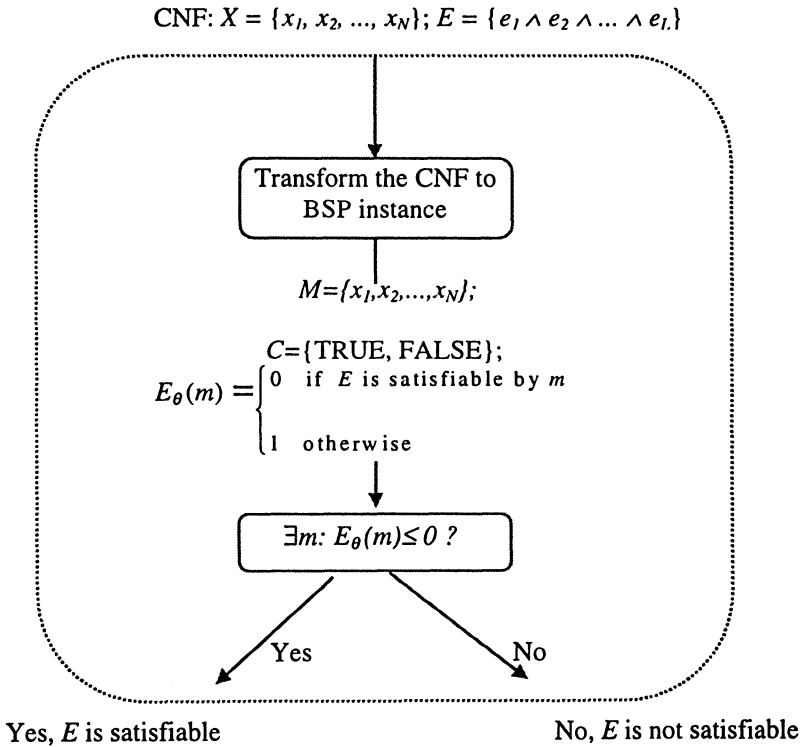


Figure 7.1 A Polynomial Transformation from SAT to BSP

7.3 THE CONSTRAINED BASIC SYNTHESIS PROBLEM (CBSP)

7.3.1 PROBLEM FORMULATION

The primitive modules and relations determine the space of possible solutions to the BSP. A designer can only generate structural descriptions that can be formed from these elements. The constraints imposed by the choice of primitive modules and relations are the cause of a fundamental trade-off between the expressiveness of the design representation scheme and the complexity of the BSP. As design representation schemes become more expressive, the space of possible solutions increases. The following constraints, based upon the specialized circumstances relevant to artifacts, will permit replacement of the formidable upper bound on Proposition 7.1 with much smaller upper bounds (but still exponential).

Here, we consider constraints with respect to three cases:

1. the possible number of assignments that each module can share, grows polynomially (of order γ) with respect to the number of primitive modules N ;
2. the number of relations of interest is bounded by some ρ ;
3. the number of assignments per relation is bounded by some v .

These constraints constitute the constrained basic synthesis problem (CBSP):

Definition 7.3 (Constrained Basic Synthesis Problem): CBSP is expressed similarly to the BSP by further considering the foregoing constraints.

Formulating the CBSP in this form enables obtaining refined upper bounds of $|S|$ (Theorems 7.2 and 7.5), and precise conditions where the CBSP can be solved in polynomial time (Proposition 7.2).

7.3.2 UNIVERSAL UPPER BOUND ON $|S|$

In this section, we obtain a *universal* upper bound of $|S|$ assuming artifacts have rigid structure as stated above (Definition 7.3). This upper bound is particularly useful when the *a priori* algorithm of solving the CBSP is unknown. In deriving the upper bound, we apply an elementary information-theoretic approach.

Let us first introduce the important quantity of the *entropy* of a random variable:

Definition 7.4: The entropy $H(X)$ of a discrete random variable X drawn according to the probability mass function $p(x)$, $x \in \Omega$, is defined by $H(X) = - \sum_{x \in \Omega} p(x) \log_2 p(x)$. We also write $H(P)$ for the above quantity. The *log* is to the base of 2 and entropy is expressed in bits.

Theorem 7.2 (Universal Upper Bound): For $v \leq 0.5N^{\gamma+1}$ and sufficiently large

$$N, |\mathfrak{S}| \leq v^{\rho} 2^{\rho N^{\gamma+1} H_0\left(\frac{v}{N^{\gamma+1}}\right)}. H_0(P) \text{ denotes the binary entropy function}$$

$$H_0(P) = -p \log_2 p - (1-p) \log_2 (1-p).$$

Proof: The number of possible assignments constituting any relation $c \in C$ is bounded by $N^{\gamma+1}$. A design solution is defined by a set of ρ relations. Consider each relation c_i contains z_i assignments ($z_i \leq v$) out of the $N^{\gamma+1}$ possible ones. Therefore the number of possible designs having this characteristic is given by

$$\prod_{i=1}^{\rho} \binom{N^{\gamma+1}}{z_i}. \text{ Hence,}$$

$$|\mathfrak{S}| \leq \underbrace{\sum_{z_1=1}^v \dots \sum_{z_{\rho}=1}^v}_{\rho} \prod_{i=1}^{\rho} \binom{N^{\gamma+1}}{z_i}. \text{ It is known that for large values of } N, \text{ the}$$

binomial coefficients satisfy $\binom{N^{\gamma+1}}{z_i} \leq \binom{N^{\gamma+1}}{v}$ (recall that $z_i \leq v$). It can also

be shown [4] that the binomial coefficients satisfy $\binom{N}{x} \leq 2^{NH_0\left(\frac{x}{N}\right)}$. Therefore, we

$$\text{obtain } \binom{N^{\gamma+1}}{v} \leq 2^{N^{\gamma+1} H_0\left(\frac{v}{N^{\gamma+1}}\right)}. \text{ Plugging into the foregoing upper bound}$$

on $|\mathfrak{S}|$ and rearranging, we obtain the required inequality. ■

Theorem 7.2 shows that the cardinality of \mathfrak{S} grows exponentially with $N^{\gamma+1}$ and with the binary entropy $H_0\left(\frac{v}{N^{\gamma+1}}\right)$. The upper bound can be controlled by a

suitable selection of N and v . The binary entropy is a concave function of the distribution, equals 0 when $p = 0$ or 1, and the maximum is obtained when $p = 0.5$. Therefore, the cardinality of \mathfrak{S} can considerably be decreased either by solving problems where v is small with respect to $N^{\gamma+1}$ (i.e., $p = \frac{v}{N^{\gamma+1}} \rightarrow 0$), or where

v approaches $N^{\gamma+1}$ (i.e., $p = \frac{v}{N^{\gamma+1}} \rightarrow 1$). Indeed, the following engineering

design heuristics, based upon the specialized circumstances relevant to artifacts, will permit replacement of the formidable universal upper bound on Theorem 7.2 with a much smaller upper bound:

- Heuristic1.* the number of relations of interest is bounded by some $\rho \ll N$;
- Heuristic2.* the number of assignments per relation is bounded by some $\nu \ll N$.

The above heuristics are now matched to a corresponding combinatorial reduction:

Proposition 7.2: For $\nu \ll N^{\gamma+1}$ and sufficiently large N , $|\mathfrak{S}|$ is bounded by: $\alpha \exp(o(N)) N^{(\gamma+1)\nu\rho}$, α is fairly small and $o(N)$ means $\frac{o(N)}{N} \xrightarrow{N \rightarrow \infty} 0$.

Proof: The proof is similar to that of Theorem 7.1, except for using the following useful approximation when needed: $\binom{N^{\gamma+1}}{\nu} \approx \frac{\exp(\nu)}{\sqrt{2\pi\nu}^{\nu+0.5}}$
 $\exp(o(N)) N^{(\gamma+1)\nu}$. Regarding the above heuristics, note that the first and third terms are fairly small relative to 2^N . ■

In summary, these heuristics imply that the total number of possible solutions that would need to be considered by the synthesis module would be quite acceptable for reasonable computational performance.

7.4 REFINED UPPER BOUND ON $|\mathfrak{S}|$

7.4.1 PROBABILISTIC DESIGN SELECTION

In this section, we tighten the universal upper bound in the last section by taking into account that the designer uses probabilistic heuristic strategy to search for solutions for the CBSP.

The nature of the information involved in the search for a design solution may be *deterministic*, by showing which designs in \mathfrak{S} are categorically inferior, or *probabilistic*, by identifying those designs having the greatest probability of solving the problem. The probabilistic decision making process is supported by many protocol studies on design [e.g., 8]. In a probabilistic framework, the designer decomposes the artifact space into the selections of $\nu\rho$ assignments. Each selection stage is labeled Node M_{ij} as depicted in Figure 7.2. Node M_{ij} is associated with a discrete random variable X_i with alphabet $\Omega = \{\lambda_{i1}, \lambda_{i2}, \dots, \lambda_{i\nu}\}$ (the set of assignments constituting relation c_i), each with a conditional probability p_{ij} of being included in a successful design solution if one exists. The choices are made out

of the $N^{\gamma+1} + 1$ possibilities (including the void assignment). If we assume that the random variables X_i are mutually independent, the designer may choose sequentially νp assignments (considering the conditional probability vectors p_i) which results in a successful design solution with high probability. The designer's problem is that he may not know which assignments will ultimately lead to a satisfying design solution. In other words, he has little or no information on the value of the p_{ij} 's. Later in Chapter 13, we develop a method for adaptive learning of these conditional probabilities p_{ij} 's.

For simplicity of analysis, we assume in the sequel that the random variables X_i are identical and independently distributed (i.i.d.) according to $p(x)$. In this case, we establish the appealing proposition that not all possible solutions in \mathfrak{S} (i.e., all the sequences, each of which has νp assignments) have the same probability of solving the CBSP, and that the probability of identifying a successful solution for the CBSP is inversely proportional to the number of assignments (νp), and to the uncertainty of the random variable X . Thus, learning about which assignments satisfy the governing requirements reduces the uncertainty of the random variable X , which in turn increases the probability of identifying a successful solution to the CBSP.

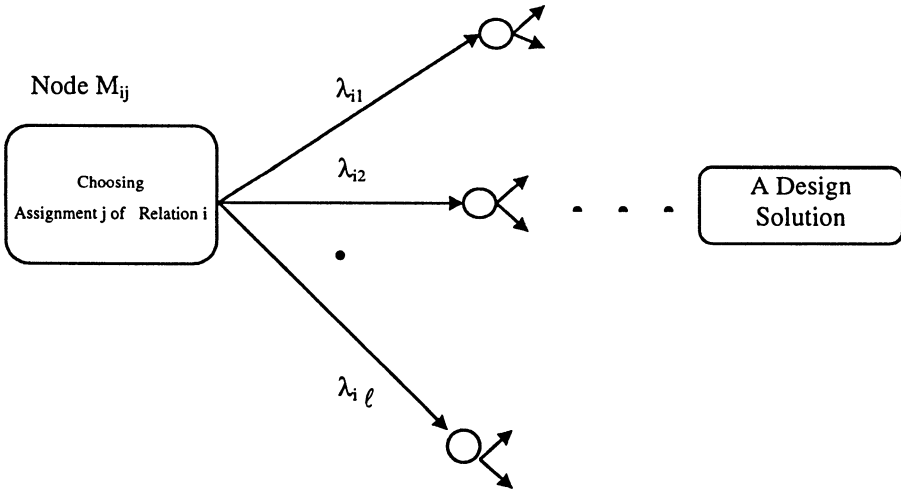


Figure 7.2 The Probabilistic Decision Making Process

7.4.2 THE ASYMPTOTIC EQUIPARTITION PROPERTY (AEP)

Let us first recall some basic results of information theory deriving from the Asymptotic Equipartition Property (AEP) which is formalized as follows:

Theorem 7.3: If X_1, X_2, \dots are i.i.d. $-p(x), x \in \Omega$, then $-\frac{1}{n} \log p(X_1, X_2, \dots, X_n) \rightarrow H(X)$ in probability

Proof: Functions of independent random variables are also independent random variables. Thus, since the X_i are i.i.d., so are $\log p(X_i)$. Hence by the weak law of large numbers,

$$-\frac{1}{n} \log p(X_1, X_2, \dots, X_n) = -\frac{1}{n} \sum_i \log p(X_i) \rightarrow -E \log p(X) \text{ in probability} = H(X), \text{ which proves the theorem.} \quad \blacksquare$$

Definition 7.5: The typical set $A_\epsilon^{(n)}$ with respect to $p(x)$ is the set of sequences $(x_1, x_2, \dots, x_n) \in \Omega^n$ with the following property:

$$2^{-n(H(X)+\epsilon)} \leq p(x_1, x_2, \dots, x_n) \leq 2^{-n(H(X)-\epsilon)}.$$

As a consequence of the AEP, it can be shown that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem 7.4:

1. If $(x_1, x_2, \dots, x_n) \in A_\epsilon^{(n)}$, then $H(X) - \epsilon \leq -\frac{1}{n} \log p(X_1, X_2, \dots, X_n) < H(X) + \epsilon$.
2. $\Pr\{ A_\epsilon^{(n)} \} > 1 - \epsilon$ for n sufficiently large.
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}$, where $|A|$ denotes the number of elements in the set A .
4. $|A_\epsilon^{(n)}| \geq (1 - \epsilon) 2^{n(H(X)-\epsilon)}$ for n sufficiently large.

Proof: The proof of property (1) is immediate from the definition of $A_\epsilon^{(n)}$. The second property follows directly from Theorem 7.3, since the probability of the event $(X_1, X_2, \dots, X_n) \in A_\epsilon^{(n)}$ tends to 1 as $n \rightarrow \infty$. Thus for any $\delta > 0$, there exists an n_0 , such that for all $n \geq n_0$, we have

$$\Pr \left\{ \left| -\frac{1}{n} \log p(X_1, X_2, \dots, X_n) - H(X) \right| < \epsilon \right\} > 1 - \delta$$

Setting $\delta = \epsilon$, we obtain the second part of the theorem. Note that we are setting ϵ for two purposes rather than using both ϵ and δ . The identification of $\delta = \epsilon$ will conveniently simplify notation later.

To prove property 3, we write

$$1 = \sum_{x \in \Omega^n} p(x) \geq \sum_{x \in A_\epsilon^{(n)}} p(x) \geq \sum_{x \in A_\epsilon^{(n)}} 2^{-n(H(X)+\epsilon)} = 2^{-n(H(X)+\epsilon)} |A_\epsilon^{(n)}|,$$

where the second inequality follows from Definition 7.5. Hence

$$|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}.$$

Finally, for sufficiently large n , $\Pr\{A_\epsilon^{(n)}\} > 1 - \epsilon$, so that

$$1 - \epsilon < \Pr\{A_\epsilon^{(n)}\} \leq \sum_{x \in A_\epsilon^{(n)}} 2^{-n(H(X)-\epsilon)} = 2^{-n(H(X)-\epsilon)} |A_\epsilon^{(n)}|,$$

where the second inequality follows from Definition 7.5. Hence,

$$|A_\epsilon^{(n)}| \geq (1 - \epsilon) 2^{n(H(X)-\epsilon)},$$

which completes the proof of the properties of $A_\epsilon^{(n)}$. ■

Thus the typical set has probability nearly 1, all elements of the typical set are nearly equiprobable, and the number of elements in the typical set is nearly $2^{nH(X)}$. Moreover, any property that is proved for the typical sequences will then be true with high probability and will determine the average behavior of a large sample. From the definition of $A_\epsilon^{(n)}$, it is clear that $A_\epsilon^{(n)}$ is a fairly small set that contains most of the probability. But from Definition 7.5 it is not clear whether it is the *smallest* such set. However, it can be shown that if the random variables X_1, X_2, \dots are i.i.d., then the typical set has essentially the same number of elements as the smallest high probability set, to first order in the exponent [4]. If the random variables X_1, X_2, \dots are not i.i.d., the direct definition of the smallest probable set may be used instead of the typical set (as shown in Chapter 13).

7.4.3 CONSEQUENCES OF THE AEP ON THE CBSP

Let us now apply the AEP to the CBSP, considering the probabilistic setting as explicated above. We let $n = \nu p$. In the context of design, the typical set $A_\epsilon^{(\nu p)}$ is interpreted as a fairly small set of design solutions that contains most of the probability of including a successful solution for the CBSP (see Figure 7.3).

Assuming that the selections of assignments are independently drawn from the probability mass function $p(x)$ and that $v\rho$ is sufficiently large, we obtain $\left|A_{\epsilon}^{(v\rho)}\right| \approx 2^{v\rho H(X)}$. Thus, we see that the size of the typical set $A_{\epsilon}^{(v\rho)}$ grows exponentially with respect to $v \cdot \rho \cdot H(X)$.

When the designer has no information (“zero information”) about which selections of assignments satisfy the governing requirements, the typical set of designs includes the entire design space. Indeed, in this case the selections are uniformly distributed over the set of possible assignments, i.e., $p(x) = \frac{1}{(N^{\gamma+1} + 1)}$. Thus, we obtain $\left|A_{\epsilon}^{(v\rho)}\right| = 2^{v\rho \log(N^{\gamma+1} + 1)} = (N^{(\gamma+1)} + 1)^{v\rho}$. As more information is obtained (by further analysis) about which assignments appear to most likely satisfy the governing requirements, the uncertainty in X can only be reduced. In this case, the entropy $H(X)$ is decreased, and the size of the typical set of designs $A_{\epsilon}^{(v\rho)}$ becomes *smaller* due to the reducible exponent $v \cdot \rho \cdot H(X)$. The designs in the modified set $A_{\epsilon}^{(v\rho)}$ have the highest (conditional) probability of solving the CBSP if one exists.

The above analysis and interpretation of the typical set suggests the following adaptive algorithm for solving the CBSP (assuming X_1, X_2, \dots are i.i.d. $\sim p(x)$). In the first stage of the algorithm, when the designer has no information (“zero information” state) about which assignments satisfy the governing requirements, the designer sets the initial representative sample, Ω^1 , to be the typical set (given ϵ) that is associated with the uniform distribution $p(x)$ for which each assignment appears with the same frequency (i.e., $p(x) = \frac{1}{(N^{\gamma+1} + 1)}$). Given such a sample, the algorithm learns more about the design’s behavior in terms of which assignments appear to satisfy the governing requirements. Accordingly, the probability mass function $p(x)$ is modified. As the new probability mass function is obtained, the algorithm constructs a new sample of candidate solutions that pertain to the modified typical set Ω^2 . This process repeats itself until some termination criterion is satisfied. Typical termination criteria include stopping when the number of solutions that have been generated exceeds a predetermined limit.

In general the random variables X_1, X_2, \dots are not i.i.d. Thus, the above analysis of the typical set is not applicable. In this case, we may use the alternate concept of the *smallest* probable set (the smallest set that contains most of the probability) in employing the adaptive algorithm for solving the CBSP. In Chapter 13, we develop such an adaptive algorithm (termed as P-learning) for designing a system that is characterized as a combination of parameters levels. The P-learning algorithm constructs a sequence of samples (populations of candidate solutions) each of which is a refined approximation of the smallest probable set.

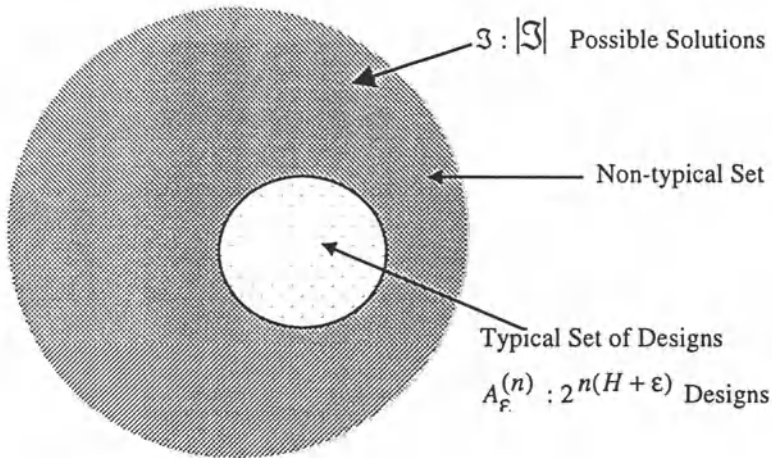


Figure 7.3 Typical Set of Design Solutions (adapted from [4])

7.5 DESIGN HEURISTICS FOR FEATURE RECOGNITION

7.5.1 GEOMETRIC MODELING

With computer graphics visualization tools we are now able to model and simulate the product being designed before building the first physical prototype. Two of the most useful computer-based visualization tools are *geometric modeling* and *computer-aided design* (CAD). These tools are software programs that use the designer's input to generate an electronic 3-D or 2-D graphic model on the computer screen. The model created by a geometric modeler or a CAD package represents a *database*. The database is a collection of data - such as the X, Y, and Z coordinates of the products' parts - having organization and structure. If the same database is shared with the manufacturing engineers, the process is called CAD/CAM, or *computer-aided-design* and *computer-aided-manufacturing*.

In designing physical devices, we use analog, iconic-descriptive, and symbolic-descriptive models to describe the unique characteristics of the product being designed. Since geometric modeling may describe the product both mathematically (symbolic) and visually (iconic), it is the most useful and comprehensive modeling technique available for developing new physical products.

Geometric models may be classified as 2-D or 3-D. A 2-D model is always a *wireframe* model. A 3-D model may be classified as wireframe, surface, or solid as briefly discussed in the following:

1. both 2-D and 3-D *wireframe models* represent objects by the edge lines, curves, and points on the surface of the object [9]. In wireframe models there are no

visible surfaces on the wireframe model; only geometric entities such as lines and curves. Wireframe models can be considered as skeletal descriptions of the product being designed. Wireframes are practical because of the speed with which they can be displayed.

2. the second type of 3-D model is *surface model*. Surface models, unlike wireframes, provide both visual and mathematical descriptions of the surface shapes of the object. Surface models are generated by placing flat and curved *patches* together to form the shell that surrounds the object. The term ‘patch’ is used by CAD modeling software developers to designate a limited region on a larger surface. Patches are mathematically defined by a curve-bounded collection of points whose coordinates are given by continuous, two-parameter functions.
3. *Solid models* are an unambiguous and informationally complete description of the object being represented. The construction procedure for solid modeling is different from that for wireframe and surface modeling. Instead of having to generate specific lines curves, and surfaces that define the object, the designer uses mathematically predefined *solid primitives*, such as blocks, cylinders, cones, wedges, spheres, and so on. The designer can define a particular primitive by specifying the desired shape, and then entering parameters such as size, position, and orientation.

7.5.2 WIREFRAME FEATURE RECOGNITION

Industrial CAD part databases consist mainly of wireframe models. These databases serve as the knowledge storage of much of the design expertness and practice. Wireframe models are considered important to the incremental redesign and betterment of future artifacts. Therefore, the design process must provide an effective mechanism for the transfer of the design knowledge captured in those wireframe databases to the synthesis stage, which is a feature-based process [10]. Informally, a feature may be thought of as a geometric configuration that has specific functional significance (e.g., slot).

In order to recognize features from wireframe models it is necessary to [10]:

1. preprocess the database to derive topological connectivity;
2. determine which connected subsets correspond to features of interest.

The first activity is not directly applicable to surface and solid models, because there already exists an explicit representation of topological connectivity for solids and surface models. However, the second activity is directly applicable to all geometric representations.

The focus of this section is upon the feasibility of feature recognition for those industrial databases whose parts contain a very large number of geometric entities (e.g., lines and arcs). This raises two fundamental questions regarding the number of geometric entities considered within a large industrial wireframe part database [10]:

1. Is the number of geometric entities that are processed to derive topological connectivity computationally intractable?
2. Is the number of connected subsets of geometric entities that are processed to match a given feature computationally intractable?

Primary attention is given to the combinatorial analysis for 2-D wireframe feature recognition (see Figure 7.4). Another domain for which the combinatorial analysis is applicable is the domain of sheet-metal parts (see Figure 7.5), where the features (e.g., filleted rectangular, slot) are cut-out features.

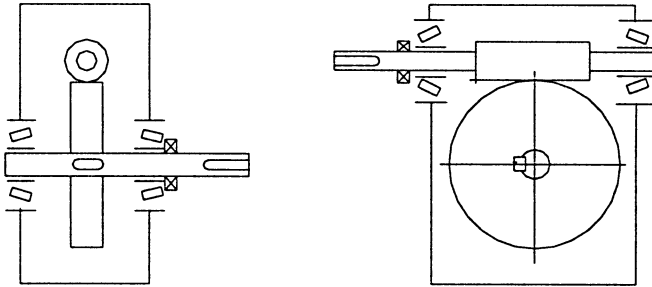


Figure 7.4 Wireframe Model of A Wormgear Reducer

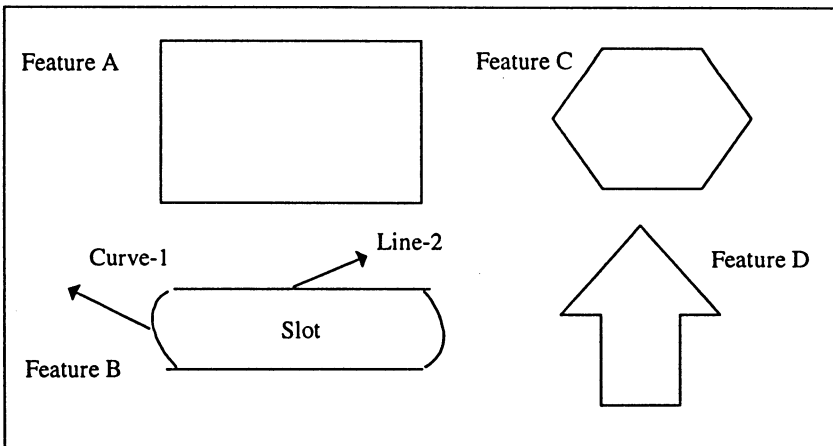


Figure 7.5 2-D Wireframe Model of Sheet-metal Part

The wireframe database in Figure 7.5 of a sheet-metal part consists of distended geometric entities, with no explicit connective information. The conventions being used are that specific geometric entities are referenced as line-2, arc-4, etc. Note that the geometric entities line-2 and line-4 both would be of entity type line and arc-2 would be of type arc. Each of the four connected subsets, labeled A-D, is a cut-out

feature. Feature B, a slot, consists of two arcs and two line segments.

Consider the problem of trying to recognize a slot from a collection of geometric entities consisting of lines and arcs. For example, a specific instance of a slot might consist of the geometric entities: arc-5, line-3, arc-4, line-2. Before this collection of two arcs and two lines could be recognized as a slot, two questions must be asked: (1) are the four entities connected?; and (2) does their connectivity match that of a slot?

The following sections apply the *guided heuristics* approach - which was applied to the *Basic Synthesis Problem* - to the analysis of the connectivity and feature recognition problems [see 10].

7.5.3 COMBINATORIAL ANALYSIS OF THE CONNECTIVITY PROBLEM

The connectivity of a slot, for example, could be specified by any cyclic permutation of the types order list (**arc, line, arc, line**), where it is understood that the consecutive elements of the list are connected end to end (with the end of the last geometric entity being connected to the beginning of the first geometric entity). Each geometric entity has a designated starting and ending point, and is called a directed entity. Thus, establishing connectivity may require to find all subsets whose connected order matches that of a feature to be recognized, and reversing the stored direction of some directed geometric entities.

The following naive analysis suggests that the combinatorial complexity of the connectivity algorithm, which is associated with preprocessing the total number of subsets of geometric entities, would be exponential, hence intractable within most modern computing environments.

Theorem 7.5 (Exponential Lower Bound): Let N denote the number of geometric entities, and \mathfrak{R} denote the set of all possible geometric subsets. Then, $|\mathfrak{R}| > 2^{N+1}$.

Proof: For a specific feature, if k is the number of entities in the feature, then it is necessary to consider those subsets of size k , namely $\binom{N}{k}$ many subsets. Each of these subsets has $k!$ many permutations. Allowing for the proper direction of individual geometric entity would increase the consideration to the following number of subsets: $\binom{N}{k} \cdot k! \cdot 2^k$. Considering all these subsets would yield a lower bound of

$$\sum_{k \leq N} \binom{N}{k} \cdot k! \cdot 2^k > \sum_{k \leq N} \binom{N}{k} \cdot 2^k > 2 \cdot \sum_{k \leq N} \binom{N}{k} = 2^{N+1}. \quad \blacksquare$$

The following engineering design heuristics, based upon domain-specific engineering knowledge, will present combinatorial upper bounds which indicate that the connectivity algorithm upon large databases will be computationally tractable.

For N geometric entities:

- Heuristic1.* the number of geometric entities per feature $\ll N$;
- Heuristic2.* the number of features of interest $\ll N$;

Next, we match each heuristic to a corresponding combinatorial reduction. Let ρ be the number of distinct classes of features of interest (a feature type is characterized by the number of geometric entities). Heuristic 2 implies that $\rho \ll N$. Let v_i be the number of entities for the i^{th} feature. Then for each $i \leq \rho$, $v_i \ll N$ as implied by Heuristic 1. Thus, the total number of subsets, $|\mathfrak{R}|$, that must be input to the preprocessing algorithm is bounded as follows:

Theorem 7.6: Let $v = \max_{i \leq \rho} v_i$, then $|\mathfrak{R}| < N^{(v+1+\frac{v}{\log_2 N})}$

Proof: Following the preceding analysis (Theorem 7.5), the total number of subsets that must be input to the preprocessing algorithm, for the i^{th} feature is $\binom{N}{v_i}$.

$$v_i! \cdot 2^{v_i} = \frac{N!}{v_i!(N-v_i)!} \cdot v_i! 2^{v_i} = \frac{N!}{v_i!(N-v_i)!} \cdot 2^{v_i} < N^{v_i} \cdot 2^{v_i} < N^v \cdot 2^v .$$

Finally, considering all the distinct classes of features would yield an upper bound of

$$|\mathfrak{R}| = \sum_{i \leq \rho} \binom{N}{v_i} \cdot v_i! \cdot 2^{v_i} < \rho \cdot N^v \cdot 2^v < N^{(v+1)} \cdot 2^v = N^{(v+1+\frac{v}{\log_2 N})} . \quad \blacksquare$$

Since the maximum number of entities per feature (i.e., v) is relatively very small relative to N , then $|\mathfrak{R}| < N^{(v+1+\frac{v}{\log_2 N})} \ll 2^N$. In summary, these heuristics imply that the total number of geometric subsets that would need to be input to the preprocessing algorithm would be quite acceptable for reasonable computational performance.

7.5.4 COMBINATORIAL ANALYSIS OF THE FEATURE RECOGNITION PROBLEM

It is the output of the connectivity algorithm, considered over all candidate subsets, that serves as input for feature recognition. A critical factor in evaluating the performance of the feature recognition module is to assess the combinatorial bound upon the number of connected subsets that may be sent to the feature recognition module. We note that once a subset is ordered and connected, those 2^k possibilities that allow for the reversal of the direction of the geometric entities have already been

fully considered and merit no further attention. The following additional design heuristic, based upon domain-specific engineering knowledge, will present a quadratic upper bound upon the number of connected subsets that may be sent to the feature recognition module:

Heuristic3. the features of interests are disjoint, and are sparsely distributed over the part.

Theorem 7.7: Let N denotes the number of geometric entities, and \mathfrak{R} denotes the number of subsets that may be sent to the feature recognition module, then $|\mathfrak{R}| < N^2$.

Proof: Regarding Heuristic 3, the feature recognition module needs only test the disjoint sets. The disjointing heuristic implies that an upper bound upon the number of disjoint connected sets of size k is N/k . Since each order list of k geometric entities has k possible starting points, it is possible to arbitrarily choose any of the k -many points to determine a particular starting point. Hence the number under consideration for any particular feature becomes $N/k \cdot k = N$. Considering all features would yield an upper bound of $|\mathfrak{R}| < \sum_{k \leq N} (N/k \cdot k) = N^2$. ■

In summary, Heuristic 3 implies that the total number of geometric subsets that would need to be input to the feature recognition module, consider all features of interest, would be a quadratic function of the total number of geometric entities. Such a bound is generally quite acceptable for reasonable computational performance.

7.6 SUMMARY

A naïve combinatorial analysis of design activities, generally applicable over all design domains, suggests that the combinatorial complexity would be exponential, hence intractable within most modern computing environments. Such a theoretical mathematical analysis ignores critical domain-specific engineering knowledge. In this chapter, it is argued that this theoretical potential for combinatorial explosion can be alleviated by combining domain-specific mechanical engineering heuristics with a detailed combinatorial analysis. This approach is termed as guided heuristics.

Two examples of such a heuristically guided combinatorial analysis are presented for the domain of basic synthesis problems (BSPs) and for the domain of feature recognition in 2-D wireframe representations:

1. the basic synthesis problem (BSP) is defined and shown to be generally intractable. Expressions for the size of the set of possible solutions are obtained,

under various conditions and engineering design heuristics. Our results are essential for developing heuristic strategies to search for optimal and suboptimal design solutions. Our main conclusion is that although expressiveness of the BSP is necessary to allow for the generation of a wide variety of designs, simply increasing the expressiveness of a design problem swamps the designer with alternatives. So, any increase in expressiveness must be accompanied by an increase in the designer's ability to control the complexity of the design space.

2. The second example of feature recognition presents combinatorial upper bounds, based on domain-specific engineering heuristics, which indicate that feature recognition upon large 2-D databases will be computationally tractable. Although primary attention is given to the combinatorial analysis for wireframe feature recognition, the given heuristics are also generally applicable to other domains such as assembly features on axially symmetric solid parts. While Heuristic 3 on Section 5.3 restricted the focus of the guided combinatorial analysis to disjoint features, it would still provide appropriate upper bounds when the percentage of interacting features is small.

In summary, the use of guided combinatorial analysis demonstrates how information from the engineering domain and the employment of engineering heuristics can be used to reduce a theoretical combinatorial explosion to a computationally tractable bound. It is expected that the general technique of augmenting combinatorial analysis by domain-specific engineering knowledge may be profitably applied to combinatorially intensive engineering domains, such as recognition of solid features, assembly planning and robotics.

APPENDIX A - THE SATISFIABILITY PROBLEM

The satisfiability (SAT in short) is expressed in terms of the following:

- A set $X = (x_1, x_2, \dots, x_N)$ of *Boolean* variables.
- *Literal* - A variable x or its negation $\sim x$.
- *Clause* over X - Defined as the disjunction of literals over X , denoted by e .
- *Expression* - Defined as a conjunction of clauses, denoted by a collection of clauses, $E = (e_1, e_2, \dots, e_M)$.
- *Satisfying truth assignment* for E - A collection E of clauses over X is *satisfiable* iff there exists true assignment for X that simultaneously satisfies all the clauses in E .

Now, SAT can be formulated as: Given a set X of variables and a collection E of clauses over X . Is there a satisfying truth assignment for E .

REFERENCES

1. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A guide to the Theory of NP-*

- Completeness*. San Francisco: W. H. Freeman and Company, 1979.
2. Krishnamoorthy, C.S., Shivakumar, H., Rajeev S. and Suresh, S., "A Knowledge-Based Systems with Generic Tools for Structural Engineering," *Structural Engineering Review*. Vol. 5 (1), 1993.
 3. Simon, H.A., *The Science of the Artificial*. Cambridge, MA: MIT Press, 1981.
 4. Cover, T. M. and J. A. Thomas, *Elements of Information Theory*. New York: Wiley and Sons, 1991.
 5. Das, S. R., D. K. Banerji, and A. K. Chattopadhyay, "On Control Memory Minimization in Microprogrammed Digital Computers," *IEEE Transactions on Computers*, Vol. C-22 (9), pp. 845-848, 1973.
 6. Grasselli, A. and U. Montanari, "On the Minimization of Read-Only Memories in Microprogrammed Digital Computers," *IEEE Transactions on Computers*, Vol. C-19 (11), pp. 1111-1114, 1970.
 7. Hayes, J. P., *Computer Architecture and Organization*. New York: McGraw-Hill, 1988.
 8. Ullman, D., T. G. Dieterich, and L. A. Stauffer, "A Model of the Mechanical Design Process Based on Empirical Data," *AI EDAM*, Vol. 2, pp. 33-52, 1988.
 9. Rodrigues, W., *The Modeling of Design Ideas*. New York: McGraw-Hill, 1992.
 10. Peters, T. J. "Mechanical Design Heuristics to Reduce the Combinatorial Complexity for Feature Recognition," *Research in Engineering Design*, 1993.

CHAPTER 8

THE MEASUREMENT OF A DESIGN STRUCTURAL AND FUNCTIONAL COMPLEXITY

The complexity of a design process or a design artifact substantially influences their performance. When evaluation of terms such as “design complexity” and its “quality” is addressed in studies, it is often performed in an *ad hoc* manner. This chapter attempts to remedy this situation by articulating two definitions of design complexity (*structural* complexity versus *functional* complexity), their associated value measures, and the relationships between them. The *structural* definition states that a design complexity is a function of its representation. Defining design complexity in the structural way provides quantitative techniques for evaluating vague terms such as ‘abstraction level’, ‘design form’s size’, and ‘designing effort’. The *functional* definition states that a design complexity is a function of its probability of successfully achieving the required specifications (functional requirements and constraints). The proposed measurable metrics provide a proper basis for evaluating each step of the design process, and accordingly recommends the direction to follow for design modification and enhancement. It also provides a framework for comparing competing artifacts (the output of a design process). Detailed examples of complexity valuation using the measures are described. The chapter concludes by discussing the scope of the measures.

8.1 INTRODUCTION

8.1.1 COMPLEXITY JUDGMENT OF ARTIFACTS AND DESIGN PROCESSES

The study of design complexity is motivated by several reasons: (1) design complexity valuation method can support the evaluation of artifacts developed in research or practice, and the determination of their relative merit. This evaluation is essential for providing feedback on research progress, such as when developing products by prototyping; (2) design complexity valuation methods can help identify good information to be used by designers or for incorporation in computer aided design systems; and (3) when an intelligent computer aided design system has several

competing modules (e.g. production rules) for solving a design task, design complexity valuation methods can identify which module to invoke for achieving the task.

In order to maintain the focus on the methodological aspects of complexity evaluation, the discussion in this chapter will refer to design complexity without reference to human design. By this we limit the discussion to the valuation of complexity embedded in intelligent computer aided design systems.

Artifact Complexity

Artifact's complexity may be said to be the exact converse of simplicity. It is generally acknowledged that the lower the artifact's complexity, ipso facto, the greater the artifact's simplicity, whereas concentration on simplicity leads to enhanced artifact's reliability and quality at lowest cost [1, 2, 3]. Simplicity is a concept used by designers and aestheticians for many centuries and remains a principle of considerable concern to them. By applying the word 'simple' to a work of art, the designer is not suggesting that the work is deficient. Simple means that the design is being reduced to the fewest possible lines, shapes and subparts, without reducing its functional requirements or violating its specifications. Thus, simple design will reduce the assembly time and product cost, as well as increase reliability by many orders of magnitude. The simplicity principle implies that if a design satisfies more than the minimum number and measure of functional requirements originally imposed, the part or process may be over-designed. Moreover, the elimination of functional requirements originally imposed will cause an incomplete design. In other words, "good" designs must be complete, yet not burdened with nonessential details. Carrol and Bellinger [1] remark that the superior design is one which encompasses the necessary operating and protective functions with the absolute minimum number of components and relations. Suh's axiomatic theory of design [4] also provides overwhelming evidence through numerous principles that support the 'simplicity' contention (e.g. "minimize the number and complexity of part surfaces for greater efficiency"). Pugh [5] remarks that the theme of simplicity may have emerged from endeavors to avoid discontinuities in systems. A discontinuity may be a bend or fork in the road, or a constriction in a pipe. Mahmoud and Pugh [6] offer a costing method for turned components produced by a variety of machines. Their costing equation is based on the number of discontinuities in the system, subsystem or component. Thus, it may be inferred that the more discontinuities you have in a system, subsystem or component, the more unreliable it will tend to be. Moreover, the number of discontinuities effects the cost due to excessive processing operations, which also has a significant impact on quality and reliability.

In software engineering, artifact descriptions range from formal languages (e.g. symbolic programming and hardware design/description languages) to very informal and visual descriptions (such as functional block diagramming and flow-diagrams). The increasing complexity of computer programs has increased the need for

objective measurements of *software complexity*. The major ways of measuring software complexity in quantitative terms fit into four categories: (1) *structured-based* measures that look to the pattern of the control flow in the software [16]; (2) *feature-based* measures that look to a selection of characteristics visible from the documentation of either the design or the source code [17]; (3) *function-based* measures that look to the pattern of input to output data correspondences [18]; and (4) *token-based* measures that look to the manner of expression of the software [15]. The *structural* complexity measures presented in Section 8.2 are based on the *token-based* methodology as introduced in [15].

Design Process Complexity

The design process was defined in Chapter 6 as an iterative scheme of decomposition and mapping between the functional and artifact domains. This iterative scheme at each level of decomposition and mapping is not unique. The evolution at each level depends on the imagination and experience of the designer, and there is no guarantee whatsoever that the same process would unfold. Thus, it is desirable to develop a systematic and generalizable framework for design process complexity valuation. Design process complexity valuation measures can aid the designer in evaluation and comparing decomposition and mapping alternatives at each level of the design process hierarchy. For example, design complexity is addressed in Suh's axiomatic design [4] by the Independence axiom: in an acceptable design, the mapping between the functional requirements and design parameters is such that each functional requirement can be satisfied without affecting any other functional requirements. At each level of decomposition and mapping, the Independence axiom is used to distinguish between acceptable and unacceptable solutions. The simplest type of design process that satisfies the Independence axiom at each level of the design process hierarchy is an uncoupled design process. In an uncoupled design process, each functional requirements can be changed without affecting any other functional requirement, which means that each functional requirement is ultimately controlled by a unique design parameter. Coupled design processes are considered highly complex and the designer should consider other mapping alternatives.

8.1.2 TWO DEFINITIONS OF DESIGN COMPLEXITY

Before discussing how design complexity may be measured, we articulate two definitions of design complexity. By "design complexity" we mean either artifact complexity or design process complexity. Both definitions assume that the study of design complexity is related to the *valuation of information* embedded in the design as captured, for example, by intelligent computer aided design systems. The valuation of information depends on what we define as information. The following definitions of design complexity dictate the type of valuation measures that can be applied:

- *Structural Design Complexity* - design complexity is a function of the design's *information content*. Information is *whatever is represented*. The design's representation may include facts, causal relations, mathematical models, etc.

Defining information in the *structural* way states that the "quantity" of information may be measured directly based on its internal structure. Design complexity is therefore a static entity.

The structural definition has several appealing properties: (1) it facilitates easy valuation of design complexity performed by simply inspecting the declarative evolutionary structure of design (with or without considering the inference mechanisms). This is much cheaper than executing behavior assessment experiments; (2) Two "amounts" of information (e.g. facts, rules, and laws included in the designer's knowledge body) could be added to yield a larger knowledge body, or knowledge could be transferred between intelligent computer aided design systems.

There are some limitations with the structural definition: (1) It detaches design from its ultimate purpose of satisfying initial specifications. Therefore, the purpose of design is not tested and, at best, can only be hypothesized from the structural measure; (2) it detaches information from its method of acquisition. Consequently, when information acquisition terminates, some meaning may be unrecoverable; and (3) it cannot explain actions that are not logical reasoning, inconsistencies, and psychological phenomena.

Applying the structural definition to evaluating artifact complexity means that if two artifacts (as captured, for example, by computer aided design databases) satisfy the required specifications, the best artifact (in terms of design complexity) is the one with the minimum information content. Thus, the complexity of an artifact may be said to be a function of its information content.

Defining design process complexity in the structural way means that if two design processes successfully achieve the required specifications, the best design process (in terms of design complexity) is the one with the minimum total information content. Thus, the complexity of a design process may be said to be a function of its information content at each level of the design process hierarchy.

There is also a definition of information as a dynamic entity:

- *Functional Design Complexity* - defining information in the functional way states that "information" is a distinct notion, independent of representation. Representations exist at the symbol level and not at the information level. In addition, information serves as the specification of what a symbol structure should be able to do. Therefore, information has a purpose, and is what a design has that allows it to attain goals.

Defining design process complexity in the functional way means that if a designer has information that one of its decisions will lead to one of its goals, than the designer will select that decision (the "principle of rationality"). Therefore, information manifest and should be evaluated functionally. This means that information can be described in terms of its operation to satisfy the goals of the

system. Alternatively, two design processes may be compared based on their output. The best design process (in terms of design complexity) is the one that yields an artifact in which its probability of successfully achieving the required specifications is maximized.

A similar “principle of rationality” is applied when evaluating artifact complexity in the functional way. Often, the behavior of an artifact is non-deterministic. Functional requirements are therefore satisfied only to a degree. Thus, a design complexity may be said to be a function of its probability of successfully achieving the required specifications. In all cases, the best artifact is the one that maximizes the probability of successfully achieving the required specifications.

8.1.3 ORGANIZATION OF THE CHAPTER

The rest of the chapter is organized as follows. Section 8.2 develops structural design complexity measures in view of the evolutionary design process developed in Chapter 6. First, some basic metrics are introduced. Second, we use the basic metrics to derive algebraic formulas for the information content of design and related *structural* complexity measures. Section 8.3 shows that the proposed complexity measures also lead to the ability to rapidly estimate the approximate total assembly time of a product, assembly efficiency, and product defect rates. Section 8.4 explores the relationships between the theoretical approach used to derive the structural complexity measures and thermodynamics. Section 8.5 presents a *functional* design complexity. Section 8.6 concludes the chapter.

8.2 STRUCTURAL DESIGN COMPLEXITY MEASURES

8.2.1 DESCRIPTION OF THE VALUATION MEASURES

In this section, we develop quantitative metrics for measuring structural design complexity. The proposed complexity measures can be used by the designer at any stage of the design process, from the conceptual design to the finality of detailed design. Following Chapter 6, the term “design” is defined as an evolutionary process. Given a description of a desired function and constraints, called specifications, the designer (or intelligent computer aided design system) provides a representation of an artifact that produces the function and satisfies the constraints. This representation, called an artifact description or artifact structure, was identified in Chapter 4 as an algebraic structure. By adaptively modifying the *design form* (the conjunction of tentative artifact and postulated specifications), from one step to the next, the designer arrives at a design solution. A design form at any particular level of abstraction (synthesis stage) is a description of an organized, mutually consistent collection of first-order and second-order constraints which a physically implemented artifact must satisfy (see Figure 8.1). The design form at a particular level of abstraction represents constraints to be met by the next lower level design. Furthermore, a higher level design may be formulated in quite fuzzy and imprecise

terms. At some later step of the design process, this same design form will have been defined in terms of more precise, more primitive constraints. [19]

Following the foregoing discussion, we define design process complexity to be a function of the “quantities” of *information content* (also termed “size”) embedded in each level of the design process (see Figure 8.1). The artifact complexity in each level of the design process may be evaluated by applying the complexity measures to the tentative artifact part. The same approach holds for the successful artifact realized in the lowest level in the design process hierarchy (Level 1 in Figure 8.1), where the artifact is described in detail. The information content has yet to be modeled. Defining information content in the structural way means that the information content should be a function of the description or representation of the design form in some *symbolic medium*, e.g. diagrams, computer languages, or first-order logic. Therefore, it may be practical to consider the design form as a computer program, i.e. an ordered string of *operators* and *operands*. An operand is a variable or a constant and an operator is an entity that can alter either the value of an operand or the order in which it is altered. For example, the artifact representation model presented in Chapter 4 is similar to the machine language of the computer, in which each instruction contains an operation code (a relation) in one segment and an operand or the address of an operand (an assignment) in the remaining section of the instruction. Henceforth we will use 'operators' and 'operands' as generic terms which include 'relations' and 'modules' as a special case. No other category of entities need be present.

We expect the *information content* of the tentative design form to vary in the course of the design process. Other “quantities” that vary in the course of the design process are the *abstraction level*, as well as the designer *effort* at arriving at the tentative design form. The aims of this section are twofold (1) to provide quantitative techniques for describing and evaluating such qualitative terms as '*information content*,' '*abstraction level*,' and '*designing effort*'; and (2) to introduce the time complexity measure, which may be used to rapidly estimate the approximate total assembly time of a product, assembly efficiency, and product defect rates.

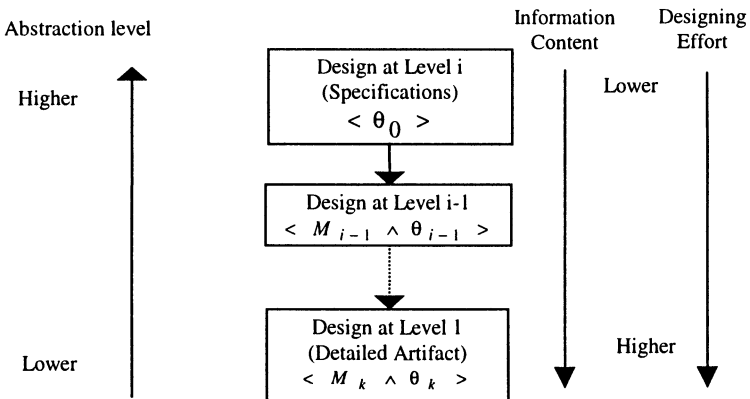


Figure 8.1 Evolutionary Design based on Abstraction Levels

8.2.2 BASIC MEASURES

Given a description of the design form in some symbolic medium (e.g. first-order logic) at a particular abstraction level, it is possible to identify all the operands and operators that the artifact representation employs (supplemented with such distinguished operators as “ \sim ”, “ \wedge ”, “ \vee ”, “()”, etc.). Similarly, it is also possible to identify all the operators and operands that constitute the specification part. Let the finite set of operators and operands (the *alphabet*) be denoted by Ω . From these simple definitions, it is possible to obtain quantitative measures for many useful properties of design processes.

Since a design form consists of an ordered string of operators and operands, and nothing else, it can be characterized by basic measures that are capable of being counted or measured:

ρ = number of unique or distinct operators appearing in the design form.

N = number of unique or distinct basic operands appearing in the design form.

$f_{1,j}$ = number of occurrences of the j th most frequently occurring operator, where $j=1,2,\dots,\rho$.

$f_{2,j}$ = number of occurrences of the j th most frequently used operand, where $j=1,2,\dots,N$.

N_1 = the total number of occurrences of the operators in the design form, $N_1 = \sum_{j=1}^{\rho} f_{1,j}$

N_2 = the total number of occurrences of the operands in the design form, $N_2 = \sum_{j=1}^{\rho} f_{2,j}$

The size of the alphabet is defined to be:

$$\eta = \rho + N \quad (1)$$

and the length of the design form to be:

$$L = N_1 + N_2 \quad (2)$$

Example 8.1: In the final stage of design of a serial binary adder unit, the design process terminates with the following successful electronic circuit (depicted in Figure 8.2). To characterize the basic measures associated with the electronic circuit, a

description of the electronic circuit in some symbolic medium needs to be considered. Here, we use the following representation in first-order predicate calculus:

$$\text{AND}(A, S_2, S_3) \wedge \text{AND}(C, B, S_1) \wedge \text{OR}(B, C, S_2) \wedge \text{OR}(S_1, S_3, \text{Output})$$

Thus, the following basic measures (constituting the alphabet) are identified: (1) signals are classified as operands; (2) logical devices are classified as operators; and (3) the distinguished symbols “ \wedge ” and “ $()$ ” are classified as operators. The counts of operators and operands are summarized in Tables 8.1 and 8.2. From the tables we see that:

$$(\rho, N_1) = (4, 12) \quad (N, N_2) = (7, 11)$$

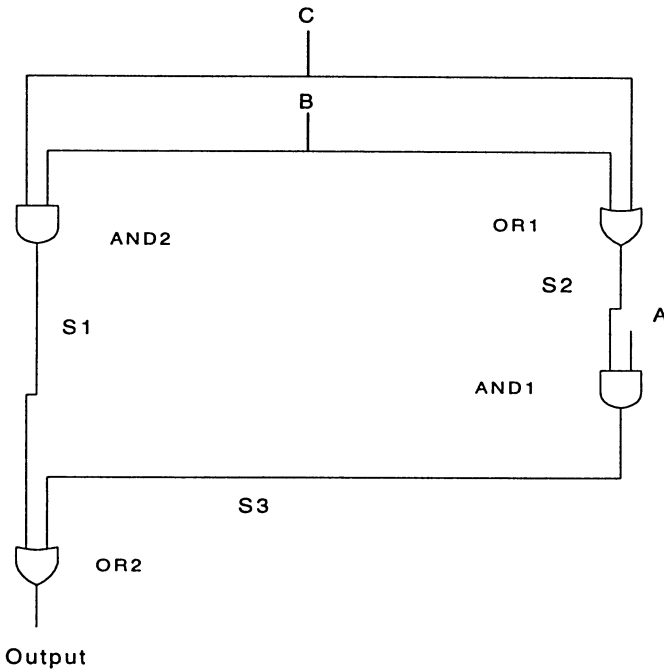


Figure 8.2 A Switching Circuit

Table 8.1 Operators related to the Switching Circuit in Figure 8.2

Relation	Count
()	4

^	3
AND	2
OR	2
	$N_1 = 11$

Table 8.2 Operands related to the Switching Circuit in Figure 8.2

Module	Count
A	1
B	2
C	2
S_1	2
S_2	2
S_3	2
Output	1
	$N_2 = 12$

8.2.3 COMPOSITE MEASURES

Based on the basic measures presented above, we develop in the remaining section three measures that may be used to express the complexity (the converse of simplicity) inherent in the design process. The proposed measures reflect such qualitative terms as *information content*, *abstraction level*, and *designing effort*.

Information Content

As mentioned earlier, the design process of any complex system proceeds on a stage-by-stage basis, and the description at each stage denotes the design form (i.e. a pair of tentative <artifact, specifications>) of the system at a particular *level of abstraction*. Note that such notions as conceptual (or preliminary) design and detailed design are subsumed within the more uniform notion of design at different abstraction levels. Note also that the nature and number of abstraction levels are entirely dependent on the designer and the type of system being designed. Given a pair of adjacent levels i , $i+1$ (where level $i+1$ is more abstract than level i), one may regard the design at level i as an implementation of the design at level $i+1$. The level i design in turn becomes a specification that is implemented by the design at level $i-1$, and so on [19].

An important characteristic of a design form at a particular abstraction level is its *size*. Whenever a given design is translated from one abstraction level to another, its size changes (see Figure 8.1). To study such changes in a quantitative way requires that the size be measurable.

A suitable metric for the size of any design form, called the *structural information content* H , can be defined as:

$$H = L \log_2 \eta = (N_1 + N_2) \log_2 (\rho + N) \quad (3)$$

The quantitative concept of the information content of a design form can be interpreted as follows: since the number of different entities in any design form is given by its alphabet, it follows that the number of bits required to provide a unique pointer (designator) for each entity must be given by $\log_2 \eta$. Using this number of bits to specify each item in the string of length L gives the information content. Thus, the information content can be interpreted as the fewest number of binary bits with which it could be represented. Later, when defining the designing effort, we will provide a more in-depth analysis of the design information content.

Information Content and Entropy

We now consider the relationship between the information content of a design form and its entropy. Following the seminal work by Shannon [7] and Wiener [8], information is defined as a reduction in uncertainty. The uncertainty preceding the occurrence of an event is usually termed *entropy*. Information theory provides a way of quantifying the *information content* received, so that the quantity of information received is equal to the reduction in entropy.

Let X_1, X_2, \dots, X_L be independent, identically distributed (i.i.d.) discrete random variables drawn according to the probability mass function $p(x) = \frac{1}{|\Omega|} = \frac{1}{\eta}$, where each variable X_i is either an operator or an operand over the alphabet Ω (the finite set of operators and operands). The probability of a sequence $(x_1, x_2, \dots, x_L) \in \Omega^L$ is $\frac{1}{|\Omega|^L} = \frac{1}{\eta^L}$. According to information theory, the

information content received from a *particular* design form with length L is equal to the *joint entropy* $H(X_1, X_2, \dots, X_L)$. The joint entropy of a sequence of discrete random variables (X_1, X_2, \dots, X_L) is defined as

$$H(X_1, X_2, \dots, X_L) = - \sum_{x \in \Omega^L} p(x_1, x_2, \dots, x_L) \log_2 p(x_1, x_2, \dots, x_L)$$

Since the probability of a sequence $(x_1, x_2, \dots, x_L) \in \Omega^L$ is $\frac{1}{|\Omega|^L} = \frac{1}{\eta^L}$, it is easy

to see that the information content associated with the particular design form is $H(X_1, X_2, \dots, X_L) = L \log_2 \eta$, in accordance with the foregoing definition.

Minimal Information Content

In translating from the higher abstract design form to the lower abstract level, we use a greater number of simpler operators and operands.

Expressing the design form, even in the highest abstraction level, would still require operators and operands. The highest (most compact) design level is formulated in ambiguous or imprecise terms. Each operator in the most compact version of the design form represents a distinct functional requirement that the required design solution is expected to satisfy. The number of operands in the most compact design form would depend upon the design problem itself, and would equal to the number of conceptually unique input and output operands of each functional requirement.

Denoting the corresponding parameters in a design's most compact representation by asterisks, it follows from Equation 3 that the minimal (also initial) information content is given by:

$$H^* = L^* \log_2 \eta^* \tag{4}$$

Note that when the initial specification includes a single functional requirement θ^0 , the design's most compact representation may be given by $\theta^0(i_1, i_2, . . . , i_{N^*})$, where $i_1, i_2, . . . , i_{N^*}$ denote the unique input and output operands. Substituting $L^* = \eta^*$ and $\rho^* = 2$ (considering the operators ' θ^0 ' and ' $()$ '), we obtain:

$$H^* = (2 + N^*) \log_2(2 + N^*) \tag{5}$$

Example 8.2: Let the initial design form (Step 1), in the binary-adder design process, be the following specification SUM(X, Y, carry-in, sum) \wedge CARRY (X, Y, carry-in, carry-out). Hence, the initial design form consists of four operators ('SUM,' 'CARRY,' ' \wedge ,' and ' $()$ ') and five operands ('X,' 'Y,' 'carry-in,' 'carry-out' and 'sum'), thus: $H^* = 13 \log_2 9 = 41.2$ bits.

Equation (5) may also be applied to evaluate the information content associated with the *artifact's* most compact representation. In this case, the artifact's most compact representation requires exactly two operators: a single operator which represents the primary function that the artifact is expected to perform, and the unique operator ' $()$ '. It also requires a list of unique input and output operands.

Example 8.3: The artifact's most compact representation associated with the electronic circuit depicted in Figure 8.2 is: BIN-ADD(X, Y, carry-in, sum, carry-out). This representation includes the operator 'BIN-ADD' which represents the primary goal, the unique operator ' $()$,' and five operands ('X,' 'Y,' 'carry-in,' 'carry-out' and 'sum'). In a lower-level form, this same design would require more operators and operands. Using the foregoing example, $H^* = 7 \log_2 7 = 19.65$ bits.

Since the minimal information content evaluates the initial design form, it may also be termed as the initial information content. Consequently, the minimal information content of a design is a single-valued function of N^* (the number of conceptually unique input or output operands). The minimal information content is the minimum possible information content associated with any given design problem in the course of the design process. It represents an absolute value against which other information contents, which are evaluated in subsequent stages, can be compared. Thus, it may be considered to be a general measure of the content of any design problem. Furthermore, it follows that H^* , unlike H , must be completely independent of the technology (e.g. methodology, design paradigm or computerized tools) or the abstraction level in which the detailed artifact is expressed.

Abstraction Level

As we could see in Figure 8.1, the level of abstraction has gone down as the information content has gone up. The information content of a given design form (at a particular abstraction level) must inversely reflect the level at which it is implemented, and the ratio of the information contents of two design forms (for the same design problem) must give the inverse of the ratio of the levels at which they have been implemented. This leads to the definition of abstraction level A as:

$$A = \frac{H^*}{H} \quad (6)$$

In the initial synthesis stage, the design form $H^* = H$; therefore, $A = 1$ (the highest level). In a lower abstraction level, the design form would have more operators and operands. Therefore, as expected, $A = H^*/H$ is less than 1 at the lower level. Furthermore, the product of information content times level will be completely abstraction level independent, because $H^* = A \cdot H$.

Example 8.4: Let us compute the artifact abstraction level associated with the electronic circuit depicted in Figure 8.2. As shown in Example 8.1, the basic measures as summarized in Tables 8.1 and 8.2 are $(\rho, N_1) = (4, 12)$, $(N, N_2) = (7, 11)$. Thus the information content associated with the electrical circuit is: $H = (N_1 + N_2) \log_2(\rho + N) = 23 \log_2(11) = 79.6$ bits. The minimal information content associated with the electrical circuit (as shown in Example 8.3) is: $H^* = 7 \log_2 7 = 19.65$ bits. Therefore, the abstraction level as defined in (6) is: $19.65/79.6 \approx 0.25$.

The Designing Effort and Time

The designing effort provides a measure for the “mental” activity required to reduce a design problem (expressed by means of initial goals) to an actual abstraction level (see Figure 8.1). The foregoing metrics and concepts provide a useful frame of reference for its quantification. The construction of a design form consists of the judicious selection of L entities (operators and operands) from a list of η entities. If a binary search method is used to select entities from the alphabet of size η , then on the average the total number of mental comparisons needed to construct a design form, is the same as the previously defined information content $H = L \log_2 \eta$. Expressing the number of “elementary mental discriminations” [9] required to make one average mental comparison as $\frac{1}{A}$, gives the effort E in units of elementary mental discriminations:

$$E = \frac{1}{A} \cdot H \tag{7}$$

Note that the *effort complexity measure* E is a linear function of the information content H (with the constant coefficient $\frac{1}{A}$). E can be also used to estimate the amount of effort required by an experienced designer to understand or comprehend the design form.

Equation 7 can be converted directly into units of time, merely by knowing the rate, S , at which the brain makes elementary mental discriminations. Stroud [9], a psychologist, defined a “moment” as the time required by the human brain to perform the most elementary discrimination. He reported that these “moments” occurred at a nearly constant rate that varies between 5 and 20 mental discriminations per second, depending on the individual [9]. While Stroud was seeking the internal processing rate of the brain, as distinct from the input/output rate, it is reassuring to note that his figures include within their range the number of frames per second which a motion picture must have to appear as a continuous picture rather than as single frames.

Provided the designer is concentrating, experienced, and has a complete specification of the design problem, the relation between time and effort is expected to be:

$$T = \left(\frac{1}{S \cdot A} \right) \cdot H = \frac{H^2}{H^* S} \tag{8}$$

Note that the *time complexity measure* T is a linear function of the information content H (with the constant coefficient $\frac{1}{S \cdot A}$).

As yet, we have not conducted protocol studies and careful experimental testing on how long it takes engineers or intelligent computer aided design systems to

synthesize an artifact; nevertheless, the timing equation is expected to show high correlation with actual designing time. The time T may also be interpreted as the time used to comprehend a design form (by reading it).

In summary, the application of effort and timing equations to design processes is based on the following assumptions:

- The designer is experienced and concentrating.
- A complete statement of the design problem (precise goals or design form) is available.
- The designer is pursuing only a single alternative, rather than considering multiple alternatives.

Note that these assumptions are embodied in the information processing postulates in psychology [10]. Moreover, the third assumption was verified previously by Ullman et al [11] by conducting protocol studies on how engineers design.

Example 8.5 (Quantitative Analysis of A Design Process): As mentioned earlier, the measurable metrics developed in this chapter may be estimated for all design forms that are encountered in the course of the design process. Thus, in working the design process, these measures will always be visible and can be continuously monitored. We shall demonstrate how this approach maps onto the evolutionary design of the mechanical fasteners presented in Chapter 6 (Table 6.1). We shall provide a self contained description of the evolutionary structure of the complexity measures underlying the design process. The initial design form (artifact and specification parts) and its complexity measures are presented in Table 8.3. Table 8.4 depicts the measures for the whole design process.

Table 8.3 Measures for the Initial Design Form of the Fastener Design Presented in Chapter 6

Design Form 0	Measures
HIGH-STRENGTH \wedge HIGH-RETRACTABILITY \wedge MEDIUM-PRECISION	$\rho^* = 1$ $N^* = 3$ $\eta^* = 4$ $N_1 = 2$ $N_2 = 3$ $L^* = 5$ $H^* = 10$ $H = 10$ $A = 1$ $E = 10$ $T = 0.55$ ($S=18$)

Table 8.4 Complexity Measures for the Fastener Design Process Presented in Chapter 6

	ρ	N	η	N_1	N_2	L	H^*	H	A	E	T
1	1	3	4	2	3	5	10	10	1	10	0.55
2	1	4	5	3	4	7	10	16.25	0.61	26.4	1.46
3	1	5	6	4	5	9	10	23.26	0.42	54.1	3.00
4	1	5	6	4	5	9	10	23.26	0.42	54.1	3.00
5	1	5	6	4	5	9	10	23.26	0.42	54.1	3.00
6	1	6	7	5	6	11	10	30.88	0.32	95.35	5.29
7	1	6	7	5	6	11	10	30.88	0.32	95.35	5.29
8	1	4	5	3	4	7	10	16.25	0.61	26.24	1.46
9	1	5	6	4	5	9	10	23.26	0.42	54.10	3.00

8.3 EVALUATING THE TOTAL ASSEMBLY TIME OF A PRODUCT

8.3.1 TOTAL ASSEMBLY TIME AND ASSEMBLY TIME MEASURE

The complexity of assembly of a product can be gauged by the time required to perform the assembly. In this section, we show that the time complexity measure T may be used to rapidly estimate the approximate total assembly time of a product. This provides a powerful analytical tool that is useful during concept development.

Following the *structural* definition of complexity, the complexity of assembly of a product is a function of its representation. In the sequel, it is suggested to use a representation that embeds the information associated with the assembly interfaces. We focus on the assembly interfaces for the following reason. As the number of mating features in an interface increases, there are additional restrictions on the orientation of the parts during assembly. As a result, the assembly time increases as the complexity of the interface grows. For example, a part with only one correct alignment orientation must have more interface dimensions and a longer assembly time than a simple cylinder that can be inserted in either axial direction into a hole.

Based on the foregoing assumption, we use the following representation in first-order predicate calculus:

$$\text{INTERFACE}(m_{i_1}, m_{i_2}) \wedge \text{INTERFACE}(m_{i_3}, m_{i_4}) \wedge \dots \wedge \text{INTERFACE}(m_{i_{k-1}}, m_{i_k})$$

where the operands m_{i_j} denote the parts of the assembly, and the operator 'INTERFACE' represents the liaisons between two separated parts. The information content H , associated with this representation, can then be computed. The most compact representation associated with the product assembly may be given as

follows:

$$\text{INTERFACE}(m_1, m_2, \dots, m_N)$$

where m_i denote the parts of the assembly. The minimal information content H^* can then be computed. Finally, the approximate total assembly time of a product as given by (8) is: $T = \frac{H^2}{H^* S}$, where S varies between 5 and 20 (we often use $S = 18$).

In Chapter 9, we inspect through an extensive statistical analysis the correlation between the time complexity measure T and the estimates of product assembly times that were derived by Boothroyd and Dewhurst in their Design for Assembly (DFA) structured methodology [12]. The correlation between the time complexity measure T and the Boothroyd and Dewhurst's estimates is found to be very close to ± 1 over a wide diversity of experiments. This demonstrates that the time complexity measure T may be used as a powerful predictive tool. By simply determining the number of interfaces and number of parts in each product concept, the approximate total assembly time can be determined with a minimum amount of analysis and without any dependence on a database. Such a tool could be used in the earliest stages of concept development to estimate the approximate total assembly times, allowing comparison of competing concepts or stimulating redesign at the time when it is easiest to make design changes.

The time complexity measure T reveals two fundamental factors that can contribute to assembly time: (1) the number of assembly operations (a subset of the set of assembly interfaces); and (2) the number of parts (a subset of the set of assembly operations). While the role of part count has long been recognized as the measure of design effectiveness [13], the method described here provides a critical missing link that relates the product assembly time to the number of operations. Without these relationships, it is impossible to accurately compare concepts that differ in the number of parts and operations.

Example 8.6 (Time Complexity Measure of A Drip Cleaner): Figure 8.3 presents a 3-D drawing of a drip cleaner, Figure 8.4 provides schematic 2-D drawings and individual names for each part of the drip cleaner, and Figure 8.5 presents the 3-D drawings and individual names for each part. Determining the 'INTERFACE' liaisons between two separated parts yields the following representation of the assembly:

$$\begin{aligned} &\text{INTERFACE}(\text{Base}, \text{ Nozzle}) \wedge \text{INTERFACE}(\text{Base}, \text{ Diaphragm}) \wedge \\ &\text{INTERFACE}(\text{Base}, \text{ Lock-Ring}) \wedge \text{INTERFACE}(\text{Base}, \text{ Membrane}) \wedge \\ &\text{INTERFACE}(\text{Base}, \text{ Cap}) \wedge \text{INTERFACE}(\text{Base}, \text{ Tube}) \wedge \text{INTERFACE}(\text{Base}, \\ &\text{Press-Ring}) \wedge \text{INTERFACE}(\text{Lock-Ring}, \text{ Diaphragm}) \wedge \text{INTERFACE}(\text{Membrane}, \\ &\text{Cap}) \end{aligned}$$

For this design form, the measurable metrics and information content are computed

as: $\rho = 3$; $N = 8$; $L = 44$; and $H = 152.21$.

To calculate H^* , we consider the most compact (highest level) representation of the drip cleaner:

INTERFACE(Base, Nozzle, Diaphragm, Lock-Ring, Cap, Tube, Press-Ring, Membrane)

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 8$; and $H^* = 33.21$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.218$; $E = 698.21$; and $T = 38.79$ seconds.

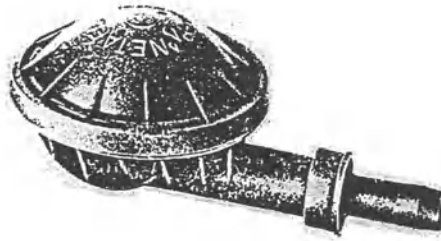


Figure 8.3 3-D Drawing of Drip-Cleaner

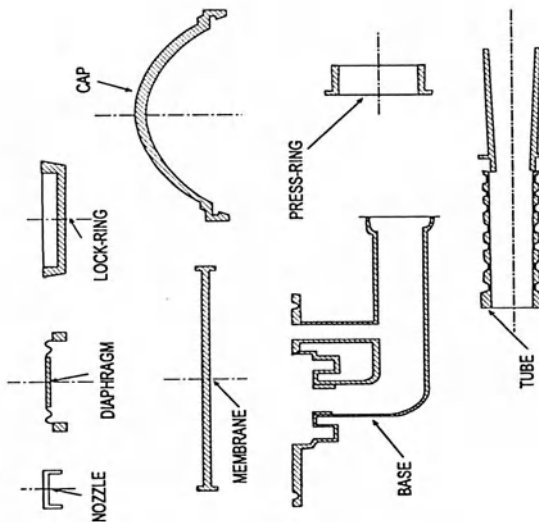


Figure 8.4 Schematic 2-D Drawings for Components of Drip-Cleaner

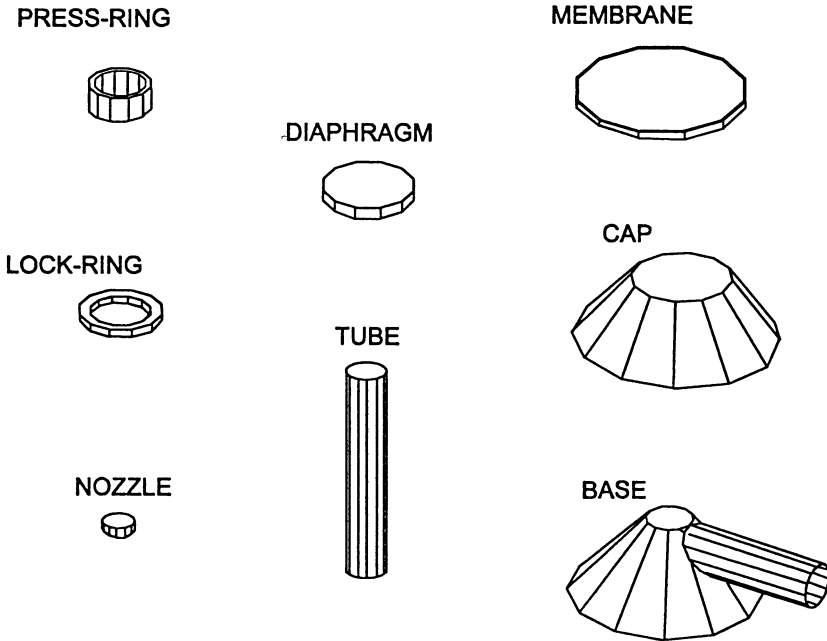
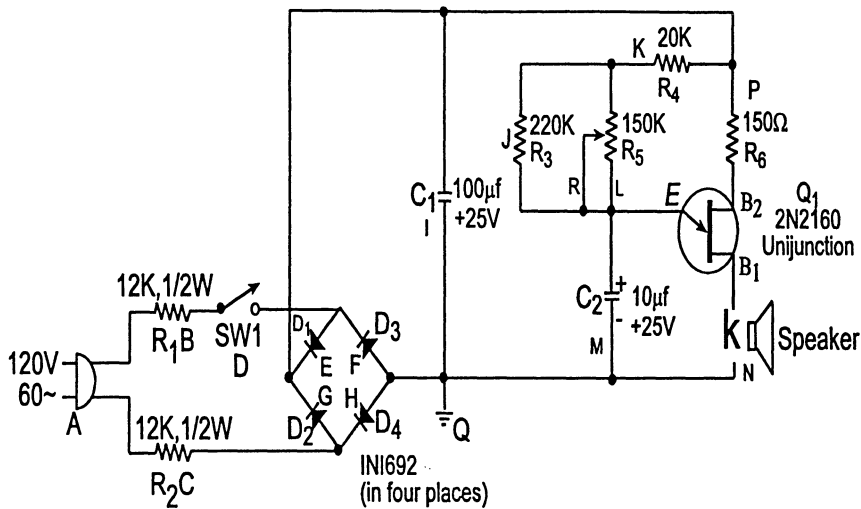


Figure 8.5 Schematic 3-D Drawings for Components of Drip-Cleaner

Example 8.7 (Time Complexity Measure of A Unijunction Transistor Metronome): The layout of the present assembly is given by Figure 8.6 [14]. The observed assembly time is given by $T = 225$ seconds. Determining the 'INTERFACE' liaisons between two separated parts yields the following representation of the assembly:

$\text{INTERFACE}(A, B) \wedge \text{INTERFACE}(A, C) \wedge \text{INTERFACE}(B, D) \wedge \text{INTERFACE}(C, G) \wedge \text{INTERFACE}(C, H) \wedge \text{INTERFACE}(D, E) \wedge \text{INTERFACE}(D, F) \wedge \text{INTERFACE}(F, Q) \wedge \text{INTERFACE}(H, Q) \wedge \text{INTERFACE}(Q, M) \wedge \text{INTERFACE}(Q, N) \wedge \text{INTERFACE}(M, N) \wedge \text{INTERFACE}(M, O) \wedge \text{INTERFACE}(N, O) \wedge \text{INTERFACE}(O, P) \wedge \text{INTERFACE}(L, M) \wedge \text{INTERFACE}(O, L) \wedge \text{INTERFACE}(K, P) \wedge \text{INTERFACE}(K, L) \wedge \text{INTERFACE}(K, J) \wedge \text{INTERFACE}(J, L) \wedge \text{INTERFACE}(I, Q) \wedge \text{INTERFACE}(I, E) \wedge \text{INTERFACE}(I, G) \wedge \text{INTERFACE}(I, K) \wedge \text{INTERFACE}(I, P)$



Parts List				
	Qty	Unit	Extended	Remarks
		price		
12-K Ω , 1/2W resistor	2	0.12	0.24	Any
20-K Ω , 150 Ω , 220K Ω , 1/4W resistors	3	0.13	0.39	Any
150-K Ω , log taper potentiometer	1	1.02	1.02	Mallory U42
Signal diode	4	0.49	1.96	Newark
SPST switch	1	0.66	0.66	Cutler-Hammer 7580K4
Capacitor, 25V, 100 μ f	1	0.81	0.81	Cornell-Dubilier Electrolytic BR100-25
Capacitor, 25V, 10 μ f	1	0.60	0.60	Cornell-Dubilier Electrolytic BR10-25
Unijunction transistor, 2N2160	1	1.49	1.49	Allied
Speaker, 3.2 Ω , 2W	1	1.85	1.85	Quam 30A05
Ground	1			

	Hours Per 100	Rate Per Hour
Finishing	18.00	\$4.60
Machine shop	7.00	5.00
Assembly	6.25	5.00
Inspection	2.80	6.20

Figure 8.6 Unijunction transistor metronome (Reproduced from [14])

For this design form, the measurable metrics and information content are computed

as: $\rho = 3$; $N = 7$; $L = 129$; and $H = 428.53$.

To calculate H^* , we consider the most compact (highest level) representation of the unijunction transistor metronome:

INTERFACE(A, B, C, ..., Q)

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 17$; and $H^* = 80.71$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.189$; $E = 2267.4$; and $T = 126$ seconds.

Example 8.8 (Time Complexity Measure of an Electrical Receptacle): The layout of the present assembly is given by Fig. 8.7, with a detailed description of its elements and a sketch of the parts involved. The time study observation as given by [14] shows that the normal cycle time (where the observed time is adjusted to a normal performance time) is found to be $T = 0.516$ minutes (31 seconds).

Let us now derive the time complexity measure T . The assembly representation is given as follows:

INTERFACE(Mount-ear, Back-plate) \wedge INTERFACE(Contact₁, Back-plate) \wedge
 INTERFACE(Contact₂, Back-plate) \wedge INTERFACE(Face-plate, Back-plate) \wedge
 INTERFACE(Face-plate, Contact₁) \wedge INTERFACE(Face-plate, Contact₂) \wedge
 INTERFACE(Screw₁, Face-plate) \wedge INTERFACE(Screw₂, Face-plate) \wedge
 INTERFACE(Screw₁, Back-plate) \wedge INTERFACE(Screw₂, Back-plate)

For this design form, the measurable metrics and information content are computed as: $\rho = 3$; $N = 7$; $L = 49$; and $H = 162.8$.

To calculate H^* , we consider the most compact (highest level) representation of the Electrical Receptacle:

INTERFACE(Mount-ear, Back-plate, Contact₁, Contact₂, Face-plate, Screw₁,
 Screw₂)

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 7$; and $H^* = 28.53$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.175$; $E = 930.28$; and $T = 51.7$ seconds.

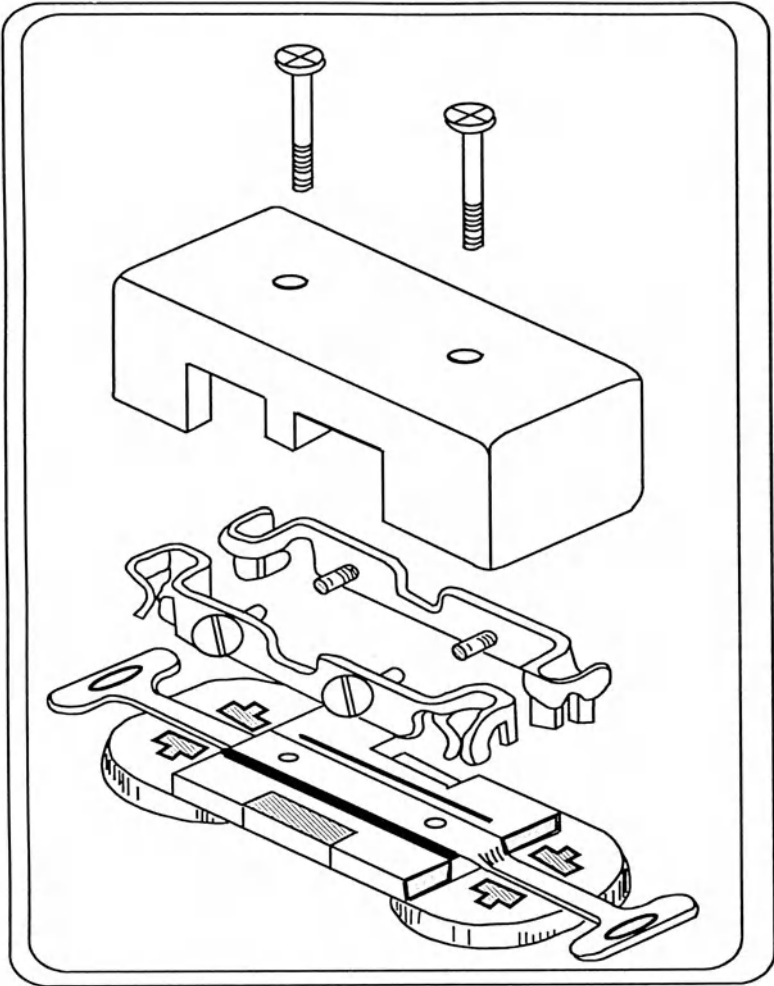


Figure 8.7 Detailed description of elements for the electrical receptacle
(Reproduced from [14])

8.3.2 ASSEMBLY DEFECT RATES AND ASSEMBLY TIME MEASURE

Even when parts satisfy defined tolerances and requirements, defects can occur during the assembly process. One source of assembly defects is *interference* between mating parts. An evaluation of several simple assemblies demonstrated that this will lead to an increased probability of assembly interference due to variations in dimensions, even for parts toleranced by the best current methods. Thus, as previously mentioned, *the assembly time increases as the complexity of the interface*

grows.

Assembly errors, such as installing a part in an incorrect position or orientation are other sources of assembly defects, which can occur during assembly even in the case of perfect parts and minimum complexity interfaces (e.g. a part may be omitted). For example, misalignment during insertion can damage mating parts that are otherwise functionally adequate. As the difficulty of the task increases, the probability of an assembly error is also likely to increase for the same level of care in the operation. Each increase in assembly time can be related to an increase in the difficulty of the assembly operation. *Thus, the probability of an assembly error should also be a function of the assembly operation time.* An analysis of eight Motorola products reflecting tens of thousands of assembly operations supported the assumption that defects would increase with assembly operation time. Fifty combinations comparing defect rates to product characteristics, such as total assembly time or number of assembly operations, were examined. Of all the combinations, the average defect per operation versus the average assembly time per operation showed the strongest linear correlation (correlation coefficient $r = 0.94$).

The finding that defects would increase with total assembly time, combined with the potential of applying the time complexity measure T to rapidly estimate the approximate total assembly time of a product (see Chapter 9), provides a method of rapidly estimating product defect rates.

8.3.3 DESIGN ASSEMBLY EFFICIENCY AND ASSEMBLY TIME MEASURE

Practitioners tend to focus on part count as the measure of design effectiveness [13]. However, as mentioned above, part count is an inadequate and potentially dangerous focus for design.

The search for a better criterion led to a study of Assembly Efficiency, a parameter introduced by Boothroyd and Dewhurst in their DFA structured methodology [12]. Assembly efficiency compares computed assembly times to an ideal but arbitrary standard. This relationship is expressed as follows:

$$EM = \frac{t_{\text{ideal}} \cdot NM}{TM} \quad (9)$$

where,

EM = the manual assembly efficiency

t_{ideal} = the "ideal" assembly time per part, suggested as 3 seconds [12]

NM = the theoretical minimum number of parts, determined as the number of parts that satisfy at least one of the following three criteria: (1) must move during

operation; (2) must be made of different material; (3) must be separate to permit assembly or disassembly

TM = the total manual assembly time in seconds

The assembly efficiency parameter can be interpreted as a measure of the potential to achieve further reduction in assembly time by redesign. The importance of the assembly efficiency parameter is recently being acknowledged. A significant relationship was observed between the defect rate in the factory assembly of several mass produced electro-mechanical products and the assembly efficiency parameter. The relationship reported by Motorola reveals a clear correspondence between the assembly efficiency rating of a given product design and defect rates encountered in production assembly. Such a relationship could provide a basis for a general, quantitative, predictive tool.

Using the time complexity measure T as an estimate of total assembly time, combined with Equation (9), we define the *assembly efficiency measure* in terms of the theoretical minimum number of parts, and the time complexity measure T :

$$EM = \frac{3 \cdot NM}{T} \tag{10}$$

We illustrate the use of the assembly efficiency measure by comparing two competing design alternatives of a pneumatic piston subassembly.

Example 8.9 (Pneumatic Piston Subassembly): Two competing concepts of a *pneumatic piston subassembly* are compared. The first alternative is presented in Figure 8.8. Determining the ‘INTERFACE’ liaisons between two separated parts yields the following representation of the assembly:

INTERFACE(Screw₁, Cover) ^ INTERFACE(Screw₂, Cover) ^
 INTERFACE(Cover, Spring) ^ INTERFACE(Spring, Piston-Stop) ^
 INTERFACE(Piston-Stop, Piston) ^ INTERFACE(Piston, Main-Block) ^
 INTERFACE(Screw₁, Main-Block) ^ INTERFACE(Screw₂, Main-Block) ^
 INTERFACE(Cover, Main-Block) ^ INTERFACE(Spring, Piston)

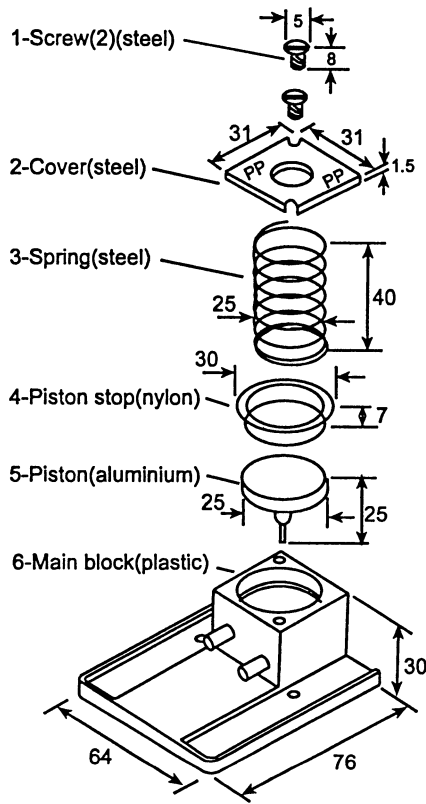


Figure 8.8 Pneumatic Piston Subassembly (Dimensions in mm). (Reproduced from [12])

For this design form, the measurable metrics and information content are computed as: $\rho = 3$; $N = 7$; $L = 49$; and $H = 162.8$.

To calculate H^* , we consider the most compact (highest level) representation of the pneumatic piston subassembly:

INTERFACE(Screw₁, Screw₂, Cover, Spring, Piston-Stop, Main-Block, Piston)

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 7$; and $H^* = 28.53$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.175$; $E = 930.28$; and $T = 51.7$ seconds.

The second alternative is presented in Figure 8.9. Determining the

‘INTERFACE’ liaisons between two separated parts yields the following representation of the assembly:

$$\text{INTERFACE}(\text{Cover \& Stop, Spring}) \wedge \text{INTERFACE}(\text{Spring, Piston}) \wedge \text{INTERFACE}(\text{Piston, Main-Block}) \wedge \text{INTERFACE}(\text{Cover \& Stop, Main-Block})$$

For this design form, the measurable metrics and information content are computed as: $\rho = 3$; $N = 4$; $L = 19$; and $H = 53.33$.

To calculate H^* , we consider the most compact (highest level) representation of the pneumatic piston subassembly:

$$\text{INTERFACE}(\text{Cover \& Stop, Spring, Piston, Main-Block})$$

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 4$; and $H^* = 15.5$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.29$; $E = 183.9$; and $T = 10.2$ seconds.

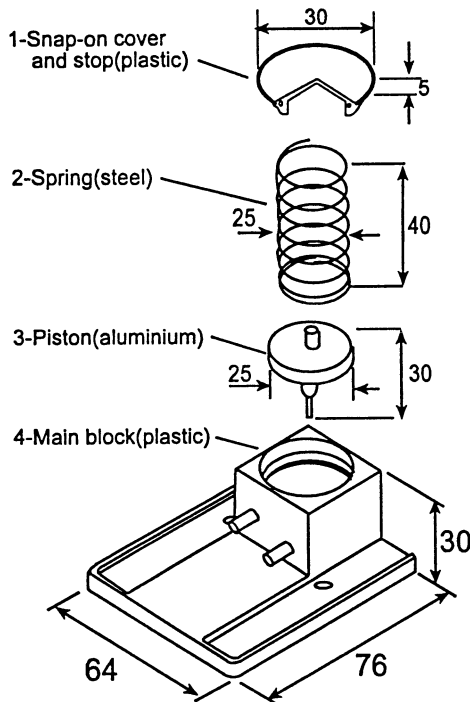


Figure 8.9 Redesign of Pneumatic Piston Subassembly (Dimensions in mm).
(Reproduced from [12])

Next we compare the two competing concepts based on their assembly efficiency measure as defined in (10). The assembly efficiency measure of the first alternative is computed as follows: $E\tilde{M} = \frac{3 \cdot NM}{T} = 0.23$, where the theoretical minimum number of parts was determined in [12] to be $NM = 4$. The assembly efficiency measure of the second alternative is computed as follows: $E\tilde{M} = \frac{3 \cdot NM}{T} = 1.17$, where the theoretical minimum number of parts was determined in [12] to be $NM = 4$. The comparison of the two competing concepts, based on their assembly efficiency measure, shows that the second alternative is preferable to the first alternative.

To conclude the example, we present the comparison of the two competing concepts by Boothroyd and Dewhurst DFA structured methodology. Tables 8.5 and 8.6 reproduce the corresponding worksheets for the subassemblies as presented by Boothroyd and Dewhurst [12]. By comparing the two products based on their manual assembly efficiency, it can be seen that the second alternative is preferable to the first alternative, as previously concluded.

Table 8.5 Worksheet for Pneumatic Piston Subassembly (Alternative 1)

1	Part ID no.	1	2	3	4	5	6		
2	Number of Times The Operation is Carried Out	1	1	1	1	1	2		
3	Manual Handling Code	30	10	10	05	23	11		
4	Manual Handling Time Per Part	1.95	1.5	1.5	1.84	2.36	1.8		
5	Manual Insertion Code	00	02	00	00	08	39		
6	Manual Insertion Time Per Part	1.5	2.5	1.5	1.5	6.5	8		
7	Operation Time (Seconds) (2) * [(4) + (6)]	3.45	4.00	3.00	3.34	8.86	19.6	42.25	TM
8	Operation Cost (Cents) 0.4 * (7)	1.38	1.60	1.20	1.34	3.54	1.84	16.9	CM
9	Theoretical Minimum Part	1	1	1	1	0	0	4	NM
		Main Block	Piston Stop	Piston	Spring	Cover	Screw	Design Efficiency $\frac{3 \cdot NM}{TM}$ $= 0.28$	

Table 8.6 Worksheet for Redesign Pneumatic Piston Subassembly (Alternative 2)

1	Part ID no.	1	2	3	4			
2	Number of Times The Operation is Carried Out	1	1	1	1			
3	Manual Handling Code	30	10	05	10			
4	Manual Handling Time Per Part	1.95	1.5	1.84	1.5			
5	Manual Insertion Code	00	02	00	03			
6	Manual Insertion Time Per Part	1.5	2.5	1.5	2.0			
7	Operation Time (Seconds) (2) * [(4) + (6)]	3.45	3.00	3.34	3.5		13.29	TM
8	Operation Cost (Cents) 0.4 * (7)	1.38	1.20	1.34	1.40		5.32	CM
9	Theoretical Minimum Part	1	1	1	1		4	NM
		Main Block	Piston	Spring	Cover and Stop		Design Efficiency $\frac{3 * NM}{TM}$ = 0.9	

The strong linear correlation between the time complexity measure T and the estimates of product assembly times that were derived by Boothroyd and Dewhurst (see Chapter 9) also supports the assumption that the manual assembly efficiency increases with the assembly efficiency measure. Considering that much less information is needed to derive the assembly efficiency measure, it makes it the more elegant and simple method.

8.4 THERMODYNAMICS AND THE DESIGN PROCESS

8.4.1 NATURAL SCIENCE AND ENGINEERING DESIGN

At present, the design process is a technical activity for which we have developed a set of ad hoc engineering metrics. Corroborative scientific research has yet to be developed. The lack of a scientific foundation is at the heart of many engineering design problems, including deficient complexity measures. There is substantial economic incentive for developing a scientific basis for design process evaluation. Following, we explore the relationship between physics (thermodynamics) and the design process, as well as the role of modeling techniques in both.

Inevitably, primitive sciences are compared to physics, which represents the standard of scientific rigor and success. Predicting the behavior of design processes can be carried out by deducing from first principles (physical theory). Thus, while it is possible to propose a biological explanation of the design process phenomenon (cf. Chapter 2), it is extremely likely that there is a physical theory of fundamental metric properties for the design process.

Some might argue that the design process as a science is philosophically different from physics as a science; because there is nothing as measurable as, for example, temperature. This sharp distinction will be seen as a question of precision rather than as a philosophical matter. Indeed - as the temperature of a body is thought of as something real, theoretically it appears as a mathematical coefficient in an equation of statistical equilibrium. There is no a priori objection to any coefficient, which appeared satisfactorily in practice (for evaluating the design process), possessing a certain degree of reality.

The remainder of this section explores the analogy between the foregoing design complexity measures and thermodynamics. This analogy also enables to generalize the proposed complexity measures.

8.4.2 THE "BALLOON MODEL"

In order to demonstrate the analogy between a general thermodynamic process and a design process, we must first state definitely what the *system* is and what the *environment* is. In thermodynamics, the system interacts with its environment through some specific thermodynamic process, starting from an initial state to a final state. During this process, energy in the form of heat (Q) and work (W) may go into or out of the system.

Let us now compute Q and W for a specific thermodynamic process. Consider a gas contained in a balloon, and assume that no heat flows into or out of the system (an irreversible adiabatic process). Let the balloon be the system, and let the blower and gas represent the environment. The foregoing physical thermodynamic process corresponds to the design process as summarized in Table 8.7. Based on this analogy, we derive the proposed design complexity measures.

Initially, the balloon is in equilibrium with the environment external to it, and has a pressure of P_i and a volume H_i . Work can be done on the balloon by compressing the gas. Consider a process whereby the system interacts with its environment and reaches a final equilibrium state characterized by a pressure P_f and a volume H_f .

Table 8.7 Thermodynamics and the Design Process

Thermodynamic Process	Design Process
Balloon	Design Form
Blower + Gas	Designer
Pressure	Abstraction Level
Volume	Information Content
Internal Energy	Designing Effort
Entropy	Information Content & Length
Power	Stroud Number
Inflationary Time	Designing Time

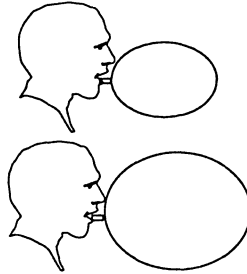


Figure 8.10 Expanding the Gas Against the Balloon

In Figure 8.10 we show the gas expanding against the balloon. The work done by the gas in displacing the balloon is given by:

$$W = \int dW = \int_{H_i}^{H_f} PdH \tag{11}$$

This integral can be graphically evaluated as the area under the curve in a *P-H* diagram, as shown for a special case in Figure 8.11.

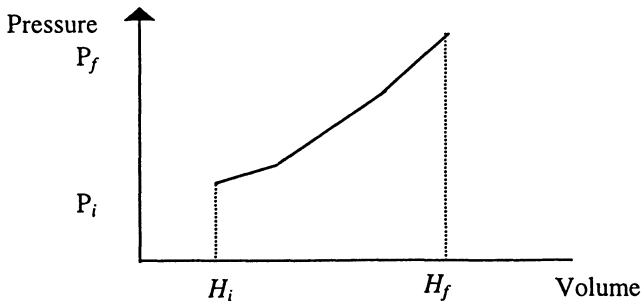


Figure 8.11 A Pressure versus Volume Diagram

There are many different ways in which the system can be taken from the initial volume H_i to the final volume H_f . However, from the first law of thermodynamics and the adiabatic process ($Q=0$), we obtain:

$$U_f - U_i = -W \quad (12)$$

where U_f , the internal energy of the system in state f , minus the internal energy of the system in state i , is simply the change in internal energy of the system. Moreover, this quantity has a definite value independent of how the system went from state i to f .

By analogy with the design process (cf. Table 8.7), we consider the special case in which the initial and final volumes $H_i (=H^*)$ and H_f represent the information content (given by Equation 3) of the initial and terminal design form, respectively.

The pressure is chosen to be $P(H) = \kappa \left(\frac{H}{H^*} \right)^\gamma$, assuming κ and γ are constants that characterize the design problem and its features. This type of expression is consistent with the *power law* and sizing model which is frequently used for estimating the cost of equipment [14]. Then,

$$W = - \int_{H^*}^{H_f} \kappa \left(\frac{H}{H^*} \right)^\gamma dH = \frac{\kappa H^*}{\gamma+1} - \frac{\kappa (H_f)^\gamma}{(H^*)^\gamma (\gamma+1)} = \frac{\kappa H^*}{\gamma+1} - \left(\frac{\kappa}{A(\gamma+1)} \right) \cdot H_f, \quad (13)$$

where A is defined as in Equation (6).

Therefore by (12) we define the internal energy as $U(H) = \left(\frac{\kappa}{A(\gamma+1)} \right) \cdot H$. Note that

W is negative when work is done on the system.

To illustrate a case, let $(\kappa, \gamma) = (2, 1)$. We obtain:

$$P(H) = 2 \left(\frac{H}{H^*} \right) = 2 A^{-1} \quad (14)$$

$$W = H^* - \frac{(H_f)^2}{H^*} \quad \text{and} \quad U_f - U_i = \frac{(H_f)^2}{H^*} - H^* \quad (15)$$

Thus $U(H) = \frac{(H)^2}{H^*}$, which is exactly the effort expression given by Equation (7).

We now formulate the design process analogy of the second law of

thermodynamics. The second law of thermodynamics can be stated, loosely, as: There exists a useful thermodynamic variable called *entropy* that is characteristic only of the state of the system, and an irreversible adiabatic thermodynamic process that starts in one equilibrium state and ends in another. This system will go in the direction that causes the entropy of the system to increase.

In statistical mechanics the quantitative relationship between entropy and disorder is given by the relation:

$$\phi = k_B \ln w \tag{16}$$

Here, k_B is Boltzmann's constant, ϕ is the entropy of the system, and w is the probability that the system will exist in the state it is in relative to all the possible states it could be in. This equation connects a thermodynamics or macroscopic (entropy) quantity, with a statistical or microscopic quantity, the probability.

Let us identify the corresponding probability w for the design process case. Here, the alphabet, η (see Equation 1), changes in the course of the design process. The probability of finding a particular operand or operator in a given design form is:

$$w^* = \frac{1}{\eta} \tag{17}$$

Thus, assuming the operands and operators are independently chosen, the probability that a given design form may be found in a certain design stage is:

$$w = \left(\frac{1}{\eta}\right)^L \tag{18}$$

where L is the length of the design form. Equation (18) coupled with Equation (16) leads to the following entropy ("k" in this equation is analogous to Boltzmann's constant):

$$\phi = -k \ln \left(\frac{1}{\eta}\right) L \tag{19}$$

Since the entropy ϕ is proportional to the length and information content (i.e. $\phi \propto L$ and $\phi \propto H$), we may identify information content and length with the qualitative idea of disorder and entropy.

Let us now consider the time involved in doing work on the balloon if the system is taken from an initial volume $H_i = H^*$ to a final volume $H_f = H$. Define the power S as the time rate at which work is done. If the power delivered by the blower (designer) is constant, then:

$$W = S \cdot T \Rightarrow T = \frac{\kappa(H_f)^{\gamma+1}}{(H^*)^{\gamma}(\gamma+1)S} - \frac{\kappa H^*}{(\gamma+1)S} \quad (20)$$

Continuing the simile with the design process, we let $(\kappa, \gamma) = (2, 1)$ and let S be the Stroud number (the rate at which the brain makes elementary mental discriminations). Hence,

$$T = \frac{(H)^2}{H^*S} - \frac{H^*}{S} \quad (21)$$

T represents the marginal time involved in changing the system from an initial volume to its final volume. Thus, we conclude that the time involved in changing the system from an initial volume $H_i = 0$ to a final volume $H_f = H$ is given by,

$$T + \frac{H^*}{S} = \frac{(H)^2}{H^*S} \quad (22)$$

Equation (20) validates the time equation derived earlier for design processes (Equation 8).

Note that there are several variations in applying the “balloon model” for the valuation of design complexity. In particular, the design complexity measures defined in Sections 8.2 and 8.3 may be generalized by choosing pairs of (κ, γ) as opposed to $(2, 1)$. The decision of which variation of (κ, γ) is used is subjective, and depends on features that characterize the design process (e.g. technology, knowledge base). For example, the complexity measures derived in Sections 8.2 and 8.3 (where the constants $(\kappa, \gamma) = (2, 1)$ are used), may be said to characterize design processes which are assisted by intelligent computer design tools. In such cases, the design activities at each step are purely mental since the designer needs to select attributes from a list of candidate solutions.

To summarize, we have argued that by analogy with thermodynamics, we may develop scientific design complexity measures. There is certainly an extensive body of theory from econometrics and related areas which can be brought to bear on the evaluation of design processes. Unfortunately, the statistical approach is a recognition that the underlying mechanisms are not understood. Therefore, we turn to the exact approach. In the exact approach, we attempt to understand - or at least quantitatively assess - the microscopic design process by applying large-scale or macroscopic formulas.

8.5 FUNCTIONAL DESIGN COMPLEXITY MEASURE

As mentioned in Section 8.1.2, the study of design complexity is related to the *valuation of information content* embedded in the design. In Section 8.1.2, we defined information in the functional way as the specification of what a symbol structure (e.g. an artifact or a design process) should be able to do. That is, information has a purpose, and is what a design has that allows it to attain goals.

Defining information content in the functional way means that the capabilities of each solution alternative may be compared with the governing set of requirements until the designer identifies the solution alternative that best satisfies the functional requirements. Without a numerical basis for comparison, however, the final selection of a design solution involving many functional requirements can only be made on a subjective or ad hoc basis. The ability to quantify how well a proposed artifact satisfies the governing requirements provides a rational means for selecting the best solution.

In this section, we define the information content of an artifact to be a function of its probability of successfully achieving the functional requirements (abbreviated as the *probability of success*). This definition is in accordance with the axiomatic theory of design that was developed by [4]. Functional information content is defined as the logarithm of the inverse of the probability of success p :

$$F = \log_2\left(\frac{1}{p}\right) \tag{23}$$

The probability of success p that relates to the satisfaction of a given functional requirement can be computed as illustrated in Figure 8.12. The “*design range*” is the *tolerance* associated with the given requirement [4]. The anticipated *response*, r , from a proposed artifact is represented as a probability density function $f(r)$. The probability of satisfying the functional requirement is depicted by the dashed area, which falls between the limits defined by the requisite tolerance (“*design range*”). Thus, the probability of success and functional information content are given by:

$$p = \text{Pr ob}[a \leq r \leq b] = \int_a^b f(r)dr \Rightarrow F = \log_2\left(\frac{1}{\int_a^b f(r)dr}\right) \tag{24}$$

As shown in Figure 8.12, the success probability can be increased by moving the mean of the response toward the desired “*design range*” and then reducing its variance. In addition, while the success probability increases, the functional information content and the artifact complexity decrease.

When there are n independent functional requirements to satisfy, the overall probability of success is given by:

$$p = \prod_{i=1}^n p_i \quad (25)$$

where p_i is the probability of satisfying the i th functional requirement as given in (24). Applying Equation (23), the total functional information content is given by the sum of the information contents associated with each functional requirement, i.e.

$$F = \log_2\left(\frac{1}{\prod_{i=1}^n p_i}\right) = \sum_{i=1}^n \log_2\left(\frac{1}{p_i}\right) = \sum_{i=1}^n F_i \quad (26)$$

Let us consider the case where the anticipated *response*, r , is represented as a uniform probability density function: $f(r) = \frac{1}{d-c}$ for $c < r < d$, and $f(r) = 0$ otherwise. The uniform probability distribution function is used in situations where the designer has no *a priori* knowledge favoring the distribution of responses except for the end points; that is, the designer does not know what the shelf life of an electrical receptacle will be but it must fall, say, between 720 and 800 hours. In the case of a uniform probability distribution, it is clear from Figure 8.13 and Equation (24) that the probability of success p is equal to the ratio of the region of overlap (called the “common range”, see [4]) between the “design range” (i.e. $[a, b]$) and the “system range” (i.e. $[c, d]$). Thus, the functional information content can be simply written as:

$$F = \log_2\left(\frac{d-c}{\max(a, c) - \min(b, d)}\right) \quad (27)$$

Example 8.10: Consider the design of a flexible manufacturing system (a detailed example is shown in Chapter 18), where the required functional requirement is represented in terms of a tolerance associated with the manufacturing system’s *production rate*, r . Let the tolerance be given by $T = \{r/r \geq 7.5\}$, and assume that the anticipated *production rate*, r , obeys the normal probability law

$$f(r) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left[\frac{r-\mu}{\sigma}\right]^2} \quad (28)$$

with mean $\mu = 8$ and standard deviation $\sigma = 1.06$. Then the probability of success p is computed as follows:

$$p = 1 - \Phi\left(\frac{7.5 - 8}{1.06}\right) = 1 - \Phi(-0.47) = \Phi(0.47) = 0.6808 \tag{29}$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{1}{2}t^2} dt$ is the standard normal probability distribution function. Thus the functional information content of the flexible manufacturing system is $F = \log_2\left(\frac{1}{p}\right) = 0.554$.

The foregoing approach is also used to define the design process functional complexity measure as the functional information content associated with the respective output solution. Thus, two design processes may be compared based on their outputs, such that the “best” design process is the one that yields an artifact in which its probability of successfully achieving the required functional requirements is maximized.

Finally, we show that the functional information content F as defined in Equation (23) is consistent with the structural information content H as defined in Equation (3). Indeed, assuming the operands and operators are independently (and sequentially) chosen, it was shown in Section 8.4 that the probability that a particular design form (a “solution”) may be found (a “functional requirement”) in a certain design stage is: $p = \left(\frac{1}{\eta}\right)^L$. Thus, the functional information content associated with

the particular design form is given by $\log_2 \frac{1}{\left(\frac{1}{\eta}\right)^L} = L \log_2 \eta$, in accordance with

the structural information content provided in Equation (3). The consistency between the structural and function complexity measures suggests that a measure based on the logarithm of the probability of success may be universal.

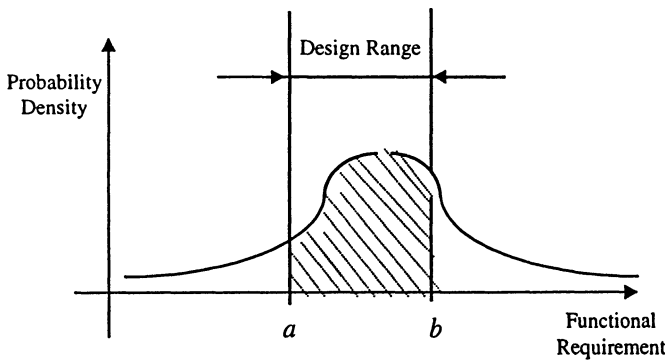


Figure 8.12 Relationship between the Probability Distribution of a System Parameter and the Designer-specified Tolerance

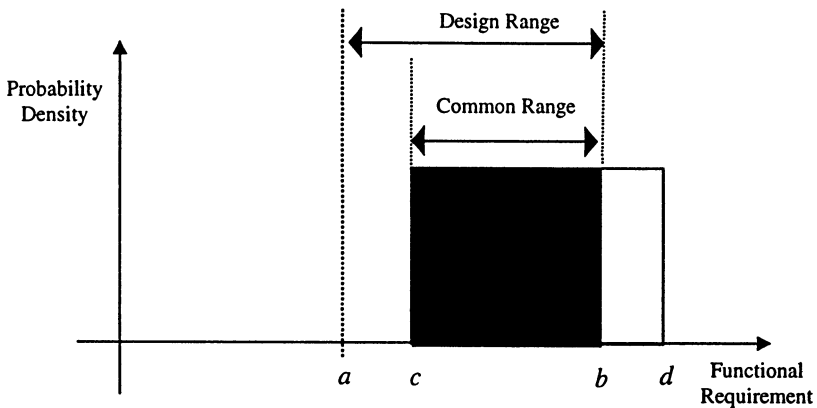


Figure 8.13 Relationship among the “Design Range,” “System Range,” and “Common Range” [4]

8.6 SUMMARY

Starting from the evolutionary model of the design process proposed in Chapter 6, we gave two definitions of design complexity (structural complexity versus functional complexity), each leading to two types of value measures.

The proposed measures enable us to evaluate the complexity of a design artifact as well as the complexity of a design process. In the course of the design process, complexity measures may be utilized by designers for comparing alternative design forms and determining which path will be most efficient. Thus, during the design process, the measurable properties will be visible and can be continuously monitored. The proposed complexity measures also lead to the ability to rapidly estimate the approximate total assembly time of a product, and the manual assembly efficiency introduced by Boothroyd and Dewhurst in their DFA structured methodology [12]. The analogy between the design process and thermodynamics as shown in Section 8.4, serves to emphasize the limited but highly useful role of science in engineering. Just as the equations of thermodynamics provide no blueprint for the design or construction of an efficient steam engine, the measures developed here tell nothing about the importance of design. In other words, they reveal how well a design form has been constructed, but they do not determine whether the design form should have been constructed in the first place. Instead, just as thermodynamics permits the engineer to calculate the maximum efficiency achievable with the optimal engine working between two specified temperatures; the design complexity measures enable the designer to calculate the “maximum efficiency” obtainable using the best possible design method working between two specified design stages.

In the next chapter, we test the hypothesis that the total assembly time of a

concept can be predicted using the time complexity measure.

REFERENCES

1. Carrol, J. T. and T. F. Bellinger, "Designing Reliability into Rubber and Plastic AC Motor Control Equipment," *IEEE Trans. Industry and General Applications*, Vol. 5 (4), pp. 455-464, 1969.
2. Crouse, R. L., "Graphic Trees Help Study of Reliability Versus Cost," *Product Engineering*, Vol. 38, pp.48-9, 1967.
3. Mihalski, J., "Design to Cost Versus - Design to Customer Requirements - Versus Design for Safe Operation: Is There a Conflict?," In *Proc. ASME, 75-SAF-2*, 1975.
4. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.
5. Pugh, S., "Load Lines: An Approach to Detail Design," *Production Engineer*, Vol. 56, pp. 15-18, 1977.
6. Mahmoud M.A.M. and S., Pugh, "The Costing of Turned Components at the Design Stage," In *Proc. Information for Designers Conf.* Southampton, pp.37-42, 1979.
7. Shannon, C. E., "A Mathematical Theory of Communication," *Bell Sys. Tech. Journal*, Vol. 27, pp. 379-423, 1948.
8. Wiener, N. *Cybernetics*, Cambridge MA: MIT Press, 1948.
9. Stroud, J. M., "The Fine Structure of Psychological Time," *Annals of New York Academy of Science*, pp. 623-631, 1966.
10. Simon, H. A. and E. A. Feigenbaum, "A Theory of the Serial Position Effect," In *Models of Thought* (Simon, H. A., ed.), Vol. 1, Yale University Press.
11. Ullman, D., T. G. Dietterich, and L. A. Stauffer, "A Model of the Mechanical Design Process Based on Empirical Data," *AI EDAM*, Vol. 2, pp. 33-52, 1988.
12. Boothroyd, G. and Dewhurst P., *Product Design for Assembly*. Wakefield, RI: Boothroyd & Dewhurst Inc, 1987.
13. Bralla, J. G., *Handbook of Product Design for Manufacturing*. New York: McGraw-Hill, 1986.
14. Ostwald, F. P., *Cost Estimation for Engineering and Management*, Englewood Cliffs, New Jersey: Prentice-Hall, 1992.
15. Halstead, M. H., *Elements of Software Science*, New York: Elsevier/North-Holland, 1977.
16. McCabe, T. J., "A Complexity Measure," *Software Engineering*, SE-2, no. 4, pp. 308-320, 1976.
17. McTap, J. L., "The Complexity of an Individual Program," In *Proceedings of the 1980 NCC*, Arlington, VA: AFIPS Press, pp. 767-771, 1980.
18. Chapin, N. "A Measure of Software Complexity," In *Proceedings of the 1979 NCC*, Arlington, VA: AFIPS Press, pp. 995-1002, 1979.
19. Dasgupta, S., "The Structure of Design Processes," In *Advances in Computers*, Vol. 28, M.C. Yovits (ed.). New York: Academic Press, pp. 1-67, 1989.

CHAPTER 9

STATISTICAL ANALYSIS OF THE TIME COMPLEXITY MEASURE

The problem of obtaining realistic and easily predictable estimations of the assembly time for a product, especially during the early stages of the design process, is usually solved utilizing Design for Assembly (DFA) methodologies which are generally based on empirical observations and experiments, and which can give good results if complete data regarding the parts used are provided to the system. In this chapter, we show that the time complexity measure presented in Chapter 8 (Equation 8.8) and the approximate total assembly time of a product, as derived from Boothroyd and Dewhurst' DFA methodology [1], are highly correlated over a wide diversity of experiments. The chapter also shows that there is logical consistency with the Barkan and Hinckley estimation method of total assembly time of a product.

9.1 INTRODUCTION

In recent years, increasing attention has been focused on design as the first step in manufacturing. While in the past the main efforts of researchers were directed to improving manufacturing equipment and processes, it is now clear that the potential benefits of this area have been widely exploited; and new, more cost effective, possibilities need to be considered.

Design is one of those areas, especially because it is now generally recognized that product design determines a great percentage of the final manufacturing cost, and that much of the manufacturing information is embedded in the product design. Therefore, since 1980, analysis techniques have been made available which can guide the designers towards products which are easy to manufacture and to assemble [2].

These techniques can be grouped into the general framework of Design for Manufacturing and Assembly methodologies (DFA and DFM), which are generally concerned with comprehending all the interactions among the various distinct processes which a manufacturing process is comprised of; and, specifically, aim to understand how product design interacts with the other components of the manufacturing system.

The work on DFA generally falls into two major areas: the assembly modeling of products and post-design analysis. Our approach falls into the latter area. Two general major principles can be recognized behind the heterogeneous DFA

methodologies [3]:

- reducing the cost of the individual piece parts manufacturing;
- reducing assembly cost and difficulty.

The various techniques generally provide the user with guidelines and systematically coded statements of design rules that have been heuristically accumulated over years of design and manufacturing experience. These techniques usually offer quantitative evaluation procedures which are derived from extensive design practice [4].

Although some of the proposed methods present extended theoretical analysis of the design process, a general scientific basis is still missing. The purpose of this chapter is to verify the hypothesis, posited in Chapter 8, that there exists a strong correlation between the time complexity measure (see Section 8.2.3) and the total assembly time of a product. We choose as a reference technique the Boothroyd-Dewhurst DFA methodology [1].

The remainder of the chapter is organized as follows: Section 9.2 provides a brief overview of three major Design for Assembly methodologies. In Section 9.3 we show that the linear correlation between the time complexity measure and the estimate of the total assembly time derived from Boothroyd-Dewhurst DFA methodology [1] is very close to ± 1 over a wide diversity of experiments.. In Section 9.4, we show that the time complexity measure falls within the 90 percent confidence interval for total assembly time as developed by Barkan and Hinckley [5]. Section 9.5 concludes the chapter.

9.2 OTHER METHODS FOR DESIGN FOR ASSEMBLY (DFA)

Even though statistics show that for many companies assembly operations account for a significant part of the production cost of the product (some authors say more than 50% [6]), it was only with the introduction of automation, in the last 1970s, that researchers began to direct their attention towards it. The first Design for Assembling methods were primarily intended to produce design suitable for automated assembly, but it became soon apparent that these methods were bringing significant savings for manual assembly, materials, and overheads.

Nowadays, DFA is considered a central element of Design for Manufacturing practice, as it aims to produce a product whose assembly cost and difficulty is minimized. Among the several methods and techniques that have been devised, the Hitachi Assemblability Evaluation Method, the Lucas DFA evaluation method, and the Boothroyd-Dewhurst DFA method (all supported by a software package) are the most popular in manufacturing industry [7].

The Hitachi DFA Method

The Assemblability Evaluation Method developed by Hitachi LTD in the late 1970s

endeavors to assess the assemblability of a product design, in the earliest possible stage of the design process, by making use of two indices:

1. the assemblability evaluation score, E , which is used to assess the design quality or the difficulty of assembly operations;
2. the estimated assembly cost ratio, K , which is used to project assembly costs relative to current assembly costs.

After having defined and categorized the motions and operations necessary to insert each part of the product, penalty points are assigned to represent the complexity of the different tasks. The penalty scores are then manipulated and combined with N (the total number of parts), and other factors influencing the elemental assembly operation, to produce the total assemblability evaluation score. The cost ratio K , which represents the total assembly operation cost of the new design divided by the total assembly operation cost of the previous design, is derived by allocating a time and cost to the basic elemental operation, and depends directly on the other index E .

The Lucas DFA Method

The Lucas method, developed in the late 1980s at the University of Hull, UK, consists of three major phases:

1. a functional analysis;
2. a handling and feeding analysis;
3. a fitting analysis.

The functional analysis addresses each component in turn and categorizes it into part type A (demanded by the design specification), and part type B (required by that particular design solution). The design efficiency is defined as $A/(A+B)$, and if the design efficiency is low a redesign must be prompted before a detailed analysis is carried on. A suggested design efficiency threshold is 60%.

In the handling/feeding analysis, where a distinction between manual and automated assembly is done, the parts are scored on the basis of size and weight, handling difficulties and orientation. A handling/feeding ratio is then calculated and compared with a recommended target value.

The final fitting analysis, based on a proposed assembly sequence, is carried out to assign to each of the individual assembly processes a fitting index, which take into account the assembly direction, alignment problems, restricted vision, the required insertion force, and eventual fixtures required. Again a fitting ratio is obtained and compared to the recommended target value.

The Boothroyd-Dewhurst DFA method

The Boothroyd-Dewhurst DFA method [1] was developed mainly at the University of Massachusetts in the early 1980s. It covers both manual and automated assembly. This method provides the user with effective tools to select the best assembling methods on the basis of the annual production volume, the payback period, the number of parts in the assembly, and the equipment costs.

The first means of improving the design is considered to be the possible reduction in the number of parts. The theoretical minimum number of parts are determined as the number of parts that satisfy at least one of the following three criteria [2]:

1. during the operation of the product, does the part move relative to all other parts already assembled?;
2. must the part be of a material that is different from those of other parts already assembled?;
3. must the part be separate from others to allow necessary assembly or disassembly of other parts?

After having recognized the fundamental parts in the design, the next problem to be addressed by the method is to ensure that the remaining parts are easy to assemble. The evaluation of the design is essentially based on the estimation of the time to complete the assembly process and its cost. In order to achieve this, a complete design-oriented classification and coding system is provided to assign time penalties to manual handling and insertion tasks. The total assembly time for a part is determined by:

1. a time penalty extracted from the coding system, that considers the design features of the part being assembled, and in particular: symmetry, size, thickness, weight, fragility, flexibility, necessity of using two hands, necessity of using grasping tools, accessibility of the assembly location, ease of operation of assembly tool, visibility of assembly location, ease of alignment and positioning during assembly, resistance to insertion, and depth of insertion;
2. the number of time that the task must be performed.

The estimated assembly time is the sum of the operation times for all the component parts.

In addition to the foregoing analytic features, Boothroyd and Dewhurst define the "Manual Assembly Efficiency" as the ideal assembling time (suggested as $3 \cdot NM$, where NM represents the theoretical minimum number of parts, and 3 seconds is the "ideal" assembly time per part) divided by the total manual assembly time in seconds.

One of the main benefits of this method is that it allows the identification of good designs and the critical examination of imperfect design solutions. Furthermore, the method is being continuously enhanced with the introduction of new methods to evaluate the cost and feasibility of a wide range of manufacturing processes. Thus, it

provides the user with effective tools to decide between different design alternatives at early stages of the design process.

9.3 RESULTS AND DISCUSSION OF THE TIME COMPLEXITY MEASURE

A major barrier to the implementation of the foregoing DFA procedures is related to the amount of time needed to acquire the methods, as well as the amount of time required to accomplish a complete analysis of a product assembly. In addition, it can be argued that the methods developed thus far neglects a “real” analysis of the design process, and that they simply constitute a black-box system which, if correctly fed with a wide range of initial data, provides the necessary results without properly clarifying the rationale behind them.

To remedy this situation, we stated in Chapter 8 the hypothesis that the total assembly time of a product can be predicted using the time complexity measure introduced in Chapter 8. In order to corroborate this hypothesis, we gathered some published design cases for which Boothroyd and Dewhurst’s estimates for the manual assembly time were available. Each product in the data set was analyzed in terms of its total number of parts and interfaces. Based on these metrics, we computed the time complexity measure using Equation (8.8), and compared it to the Boothroyd and Dewhurst’s estimate.

In Table 9.1, we show the list of artifacts, the total manual assembly time derived by the Boothroyd and Dewhurst’s DFA method [8-12], and the time complexity measure introduced in Chapter 8 (an example is given in Appendix A).

We performed a linear regression analysis to identify the correlation between Boothroyd and Dewhurst’s DFA estimates and the time complexity measure. The following regression line was obtained:

$$T = 2.74 \cdot TM - 86.82 \quad (1)$$

where TM is the total manual assembly time in seconds, and T is the time complexity measure. A strong linear correlation (correlation coefficient $r = 0.92$) was found between the total manual assembly time and the time complexity measure. The highly correlated plot is presented in Figure 9.1. Based on the regression line in Equation (1), the total manual assembly time of a product can be estimated as follows:

$$TM = 0.36 \cdot T + 31.7 \quad (2)$$

The additive constant in Equation (1) is negative, which suggests that the values near the origin are not significant. If we consider only the upper 95% of the analyzed values, the linear regression line becomes:

$$T = 3.17 \cdot TM - 26.27 \quad (3)$$

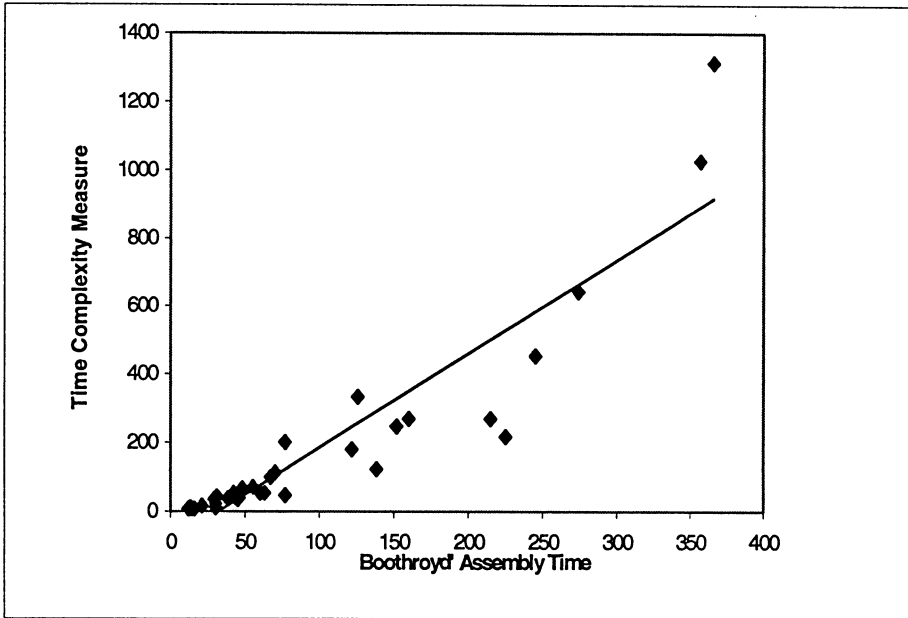


Figure 9.1 The Total Manual Assembly Time versus the Time Complexity Measure

We also reviewed the existing data to verify if other factors influence the total manual assembly time. Several combinations comparing total assembly times to product characteristics, such as number of assembly operations, and the number of parts. Of all the combinations, the time complexity measure versus the total manual assembly time derived by Boothroyd and Dewhurst's DFA methodology showed the strongest linear correlation. These results corroborate our hypothesis that the time complexity measure may be used as a powerful predictive tool in this application.

We can also observe that, for all the redesign cases analyzed, the improved design presents a better value of the time complexity measure, i.e. the value of the time complexity measure T is smaller than the one calculated for the original artifact (prior to redesign). This result demonstrates that the time complexity measure allows comparison of competing concepts, or stimulating redesign at the time when it is easiest to make design changes. If the new design has a lower time complexity measure, it may render the artifact most suitable. Various scenarios of design changes may be considered on the basis of the time complexity measure along with other considerations that are related to the functional and manufacturing requirements.

Table 9.1 The List of Design Cases

	Boothroyd-Dewhurst Assembly Time	Time Complexity Measure
motor assembly	160	269
motor assembly redesign	46	39.74
controller assembly	215	269
controller assembly redesign	63	53.7
power plug	43.2	40.5
riser panel	152	247
riser panel redesign	70	113
power saw	274	642
power saw redesign	126	333
door latch mechanism	366	1313
door latch mechanism redesign	77	201
spindle assembly	245	453
housing assembly	60	53.7
gear assembly	45	34.4
piston assembly	30	11.2
link assembly	14	7.2
piston ₂ assembly	38	38
power saw assembly	67	99.5
cylinder-base assembly	357	1025
battery charger	55	70.7
housing ₂ assembly	29	34.4
gear assembly	16	7.2
piston assembly	12	7.2
piston assembly	21	15.9
power saw assembly	48	67.2
pneumatic piston	42.2	53.7
pneumatic piston redesign	13.3	11.3
terminal subassembly	30	22.9
riser panel assembly	122	181
unijunction transistor metronome	225	217
electrical receptacle	31	43.3
metal frame assembly	138.5	123
controller assembly	77	47.1

9.4 THE BARKAN AND HINCKLEY ESTIMATION METHOD

Barkan and Hinckley [5] postulate that in any design, the set of manual assembly times per operation follows a Pareto distribution (or Zipf distribution). The Pareto

distribution is a model for phenomena where the likelihood of an event decreases as its magnitude increases. For example, the higher the salary, the fewer the number who will have that salary.

A Pareto distribution of assembly times per operation for a typical product illustrates that the shortest assembly times are the most likely to occur. Comparing a particular product to a similar product having roughly ten times as many operations, approximately the same relative fraction of operations would fall within the same time steps. To illustrate, let a motor drive assembly consists of 20 assembly operations, and assume that 10 of the 20 operations require less than ten seconds to complete. For the same Pareto distribution of assembly times per operation, in a motor drive assembly having 100 operations, approximately 50 of those operations would take less than ten seconds each to complete.

The probability mass function of the Pareto or Zipf distribution is given by the following equation:

$$Prob\{X = k\} = \frac{C}{k^{\alpha+1}} \text{ for } k = 1, 2, 3, \dots \quad (4)$$

This equation, which gives the probability that X will have a value of k , is based on two constants, α and C . Since the sum of the probabilities must equal one, the value of C is fixed by α . For distributions such as assembly time, k can represent bins of any uniform size. For example, with $\alpha = 1.225$ ($C \cong 0.6779$), the probability that an observation would fall in the first bin is 0.6779, in the second bin would be 0.145, and so forth. Given 30 observations with a time bin size of five seconds, we would expect $0.6779 \cdot 30 \cong 20$ observations in the five-second bin and $0.145 \cdot 30 \cong 4$ observations in the ten-second bin.

Statistical methods have confirmed [5] that assembly operation times follow a Pareto distribution described by two adjustable coefficients. At the same time, the statistical methods reject at highly significant levels the possibility that assembly operation times are normally distributed.

Having identified the Pareto probability distribution as an appropriate distribution for assembly operation times, standard methods are available for estimating the cumulative outcome of a series of random assembly operation times (random trials or selections) from the Pareto distribution. Using these techniques, a probabilistic relationship between total assembly time and the number of assembly operations based on $\alpha = 1.55$ and a time step increment of 4.4 seconds has been found to be consistent with the results obtained for 228 diverse assemblies.

Using the Pareto distribution of assembly times, it has also been shown that the approximate total assembly time of a product can be bounded -- based on a 90 percent confidence interval -- as a function of the number of assembly operations (N_A) as follows:

$$TM_{min} = 3 \cdot (0.81 + 2.07 \cdot N_A - 2 \cdot (N_A)^{0.5}) \quad (5)$$

$$TM_{max} = 3 \cdot (-5.72 + 3.18 \cdot N_A + 7.6 \cdot (N_A)^{0.5}) \quad (6)$$

A preliminary analysis of the results in Table 9.1 shows that nearly 80 percent of the estimated manual assembly times determined by Equation (2) fall within the above bounds. Thus, our approach is found to be consistent with the hypothesis that the set of manual assembly times per operation follows a Pareto distribution.

9.5 CONCLUSIONS

Design for Assembly (DFA) is directly related and inextricably linked to designing for cost [1]. The main aims of DFA are to minimize components, assembly cost and development cycles; and to enable higher-quality products to be made. The importance of Design for Assembly as a fundamental feature of every Design for Manufacturing (DFM) strategy is now generally recognized, and, therefore, different methodologies have been developed to systematically lead the designer to a product that is easy to assemble.

The user of a DFA methodology needs a detailed knowledge of manual product assembly times, since assembly times can be considered one of the most important parameter for the evaluation of different design alternatives. Unfortunately, this information is often unknown because assembly times have not been fully specified in the early conceptual stages. In this chapter, we tested a new approach to the estimation of manual assembly time of a product, based on a set of design measures developed to evaluate design complexity by means of simple and rational principles.

The results show that the correlation between the time complexity measure and the estimation of the total manual assembly time derived from Boothroyd-Dewhurst DFA methodology is found to be very close to ± 1 over a wide diversity of experiments. In addition, we show that the estimated total assembly time, using the time complexity measure, falls within the 90 percent confidence interval for total assembly time as developed by Barkan and Hinckley (derived from the Pareto distribution). Thus, our approach (i.e. applying the complexity measures introduced in Chapter 8) has been found to be consistent with other design knowledge, and may be advanced to the status of a theory.

Since the time complexity measure and the total manual assembly time are found to be highly correlated over a wide diversity of experiments, the time complexity measures (and the related measures introduced in Chapter 8) could provide a basis for a general, quantitative, predictive tool. Such a tool could be used in the earliest stages of concept development to evaluate the product complexity, and the approximate total assembly time. It also allows the comparison of competing concepts or stimulating redesign at the time when it is easiest to make design changes. Rapid estimation of the total assembly time may be supported by imbedding the evaluation method within a computer aided design (CAD) tool, which can interactively provide at the conceptual stage of the design process the information regarding the product's layout (e.g. interfaces, and parts).

In developing or selecting a DFA procedure we often encounter a tradeoff: while accurate and precise results are requested, the difficulty and the amount of data (and

consequently the time to gather and analyze them) required to obtain them increases. While the Boothroyd and Dewhurst's DFA methodology represents an exhaustive and effective tool to optimize the design, it requires a lot of data and computational effort to complete the analysis. Since the only data required to compute the time complexity measure are the number of parts and the product liaison diagram, the proposed measure can partially avoid the above tradeoff by providing relatively accurate results using few data and incurring low computational efforts.

If the strong correlation between the time complexity measure and the total assembly time of a product is further corroborated, the following conclusions may be drawn: (1) part count alone is not an adequate basis for defining design simplicity or predicting conformance quality. Instead, a superior design criterion is: *minimize and simplify* assembly operations. This tends to reduce part counts and simplify part interfaces while avoiding assembly complexity that may be introduced to achieve part count reduction; (2) the time complexity measure increases more than linearly with respect to the number of operations in an assembly. Thus, in a product that requires more assembly operations than another product, it is likely that it will have some assembly operations that are more time consuming; (3) the time complexity measure may be used to rapidly estimate the manual assembly efficiency introduced by Boothroyd and Dewhurst in their DFA structured methodology.

The method described here can be extended in a number of ways: (1) the design examples examined in this chapter were mainly drawn from electromechanical domains. Further statistical analysis should be conducted for artifacts that belongs to other domains (e.g. Printed Circuit Board assembly) in order to verify the strong linear correlation between the time complexity measure and total manual assembly time; (2) the proposed method does not consider the complexity of the single part or the complexity of the single assembly operation, but only the overall time complexity of the product. The features of the single part that affect the assembly operation (e.g. size, symmetry, weight, fragility, thickness, etc.) should be embodied in the time complexity measure. In particular, in developing the time complexity measure (see Chapter 8), we assumed that every single interference between two parts contributes in the same way to the length (L) and information content (H) of the design form. It may be possible to introduce features that depend on the complexity of the two mating parts. Applying these features, we assign different weights to different interfaces so that higher weights correspond to difficult assembly operations. The adjusted length of the design form will be:

$$\hat{L} = \sum_{i=1}^M w_i 4 + (M - 1) \quad (7)$$

where '4' denotes the total number of operands and operators required for the representation of one interface, M denotes the total number of interfaces, and w_i denotes the weight associated with the i th interface.

Finally, the adjusted information content will be:

$$\hat{H} = \hat{L} \log_2 (\rho + N) \tag{8}$$

APPENDIX A - TIME COMPLEXITY MEASURE OF A MOTOR DRIVE ASSEMBLY

Figure 9.2 presents a redesign of a motor drive assembly. Determining the ‘INTERFACE’ liaisons between two separated parts yields the following representation of the assembly:

INTERFACE(Sensor, Base) \wedge INTERFACE(Set-Screw, Base) \wedge INTERFACE(Set-Screw, Sensor) \wedge INTERFACE(Motor-Screw, Motor) \wedge INTERFACE(Motor-Screw, Motor) \wedge INTERFACE(Motor-Screw, Base) \wedge INTERFACE(Motor-Screw, Base) \wedge INTERFACE(Cover, Base) \wedge INTERFACE(Motor, Cover) \wedge INTERFACE(Motor, Base)

For this design form, the measurable metrics and information content are computed as: $\rho = 3$; $N = 7$; $L = 43$; and $H = 142.84$.

To calculate H^* , we consider the most compact (highest level) representation of the motor drive assembly:

MOTOR-DRIVE(Motor Screw₁, Motor Screw₂, Base, Cover, Sensor, Set Screw, Motor)

For this design form, the measurable metrics and minimal information content are computed as: $\rho^* = 2$; $N^* = 7$; and $H^* = 28.52$. Finally, the abstraction level, effort, and time complexity measure are respectively given by: $A = 0.199$; $E = 715.4$; and $T = 39.74$ seconds.

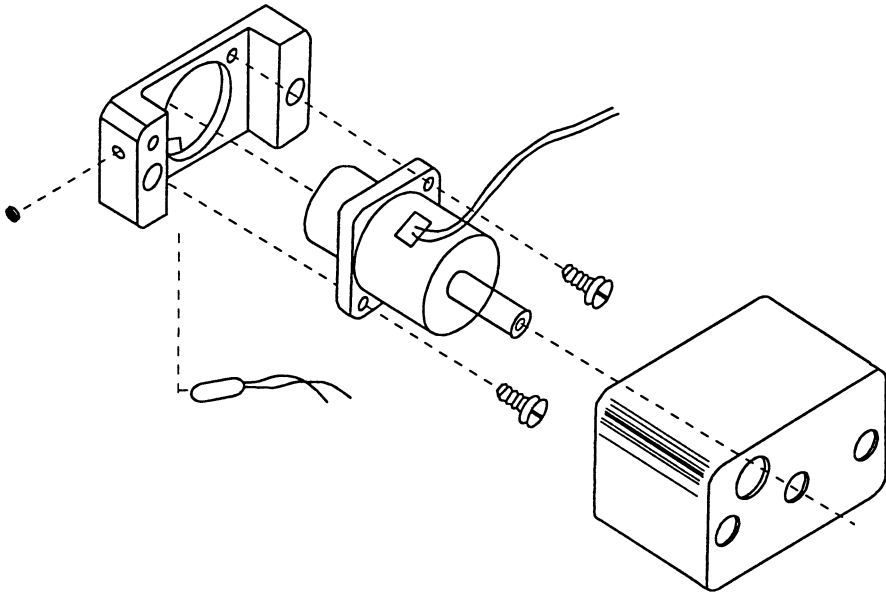


Figure 9.2 Motor-Drive Assembly Redesign

REFERENCES

1. Boothroyd, G. and Dewhurst P., *Product Design for Assembly*. Wakefield, RI: Boothroyd & Dewhurst Inc, 1987.
2. Boothroyd, G., "Product Design for Manufacture and Assembly," *Computer Aided Design*, Vol. 7, 1994.
3. Pearson, A. S., "Investigating the Dimensions of Design Decision Making Through Product Archeology," *Technical Report*, MIT, 1992.
4. Redford, A. and Chal, J., *Design for Assembly - Principles and Practice*. McGraw-Hill, 1994.
5. Barkan, P., and Hinckley, C. M., "The Benefits and Limitations of Structured Design Methodologies," *Manufacturing Review*, Vol. 8, No. 3, 1993.
6. Bhattacharya, "Comparative Analysis and Applications of Various Design for Assembly Methodologies to the Design of Electro-Mechanical Products," Florida Atlantic University, Boca Raton, Florida, December, 1992.
7. Leany, P. G., and Wittenberg, G., "Design for Assembling," *Assembly Automation*, Vol. 2, 1992.
8. Boothroyd, G., *Assembly Automation and Product Design*. Marcel Dekker Inc., New York, 1992.
9. Fujita, T., and Boothroyd, G., "Data Sheet and Case Study for Manual Assembly," *Report # 16*, Department of Mechanical Engineering, University of Massachusetts, Amherst, April, 1992.
10. De Lisson, W. A., and Boothroyd, G., "Analysis of Product Designs for Ease of Manual Assembly - A Systematic Approach," *Report # 7*, Department of Mechanical Engineering, University of Massachusetts. Amherst, May, 1992.
11. Boothroyd, G., "Design for Assembly - The Key to Design for Manufacture," *Report # 9*, Department of Industrial and Manufacturing Engineering, Kingston, Rhode Island, January, 1987.
12. Porter, C. A., and Knight, W. A., "Design for Quality," *Report # 71*, Department of Industrial and Manufacturing Engineering, Kingston, Rhode Island, February, 1987.

PART THREE

ALGORITHMIC AND HEURISTIC METHODS FOR DESIGN DECISION SUPPORT

CHAPTER 10

INTELLIGENT ADVISORY TOOL FOR DESIGN DECOMPOSITION

In this chapter, the design search problem of finding a finite sequence of production rules that begins with initial specifications and ends with an acceptable solution (as introduced in Chapter 6), is represented by an AND/OR search tree. A design search algorithm, which is based on a set of production rules and the AND/OR tree representation, is used to search for a consistent (i.e., physically realizable) design solution. A prototype system that was developed to implement the framework is discussed.

10.1 INTRODUCTION

According to the model presented in Chapter 6, the design process problem is characterized by the initial process state, the set of operators, and the accepting (goal) process states. The initial specifications represent the initial process state, a set of production rules for the decomposition of specifications is the set of operators, and design solutions are the accepting process states. In design, the initial process state is known *a priori* while the accepting state is to be determined. An operator transforms the given process state into a different process state. The task is to find a sequence of operators (production rules) that will lead to an accepting process state. The set of all process states that can be reached by applying production rules is called the *state space*. Those states in the state space that are accepting constitute the *solution space*.

Depending on the initial specifications, there may exist a large number of alternative sequences of production rules that lead to accepting states. Thus also a large number of alternative design solutions may exist. Theorem 6.3 teaches us that it is inherently difficult (NP-hard) to identify the sequence of production rules used by the design solving process before reaching an accepting state. This theorem concludes that design problem solving, particularly with its large volume of selection information, requires the aid of a systematic selection process guideline. In this chapter, a branching AND/OR tree search mechanism is presented jointly with an intelligent advisory tool to guide the design process search.

In Section 10.2, the AND/OR tree representation of design is discussed. A methodology for guiding the AND/OR search tree in the state space is presented in Section 10.3. In Section 10.4, we discuss a prototype system that has been developed

as an illustration of the practical application of our framework. Conclusions are drawn in Section 10.5.

10.2 AND/OR TREE REPRESENTATION OF DESIGN

It is helpful to display the type-2 design process introduced in Chapter 6 as an AND/OR tree. An AND/OR tree is useful for representing the behavior of a rule-based design system that works by problem decomposition; i.e., by decomposing high-level specifications into sub-specifications, which may have their own sub-specifications, and so on. Thus, to achieve high-level specifications, sub-specifications should be derived. There are two types of nodes in an AND/OR tree: OR nodes and AND nodes. An OR node is satisfied if one sub-specification (child node) is satisfied, and an AND node is satisfied if all the sub-specifications are satisfied in order for the node to be satisfied. These and other AND/OR clause representations are depicted in Table 10.1. Let us consider the following:

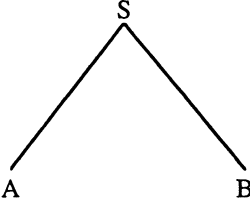
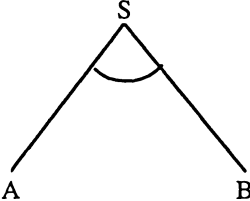
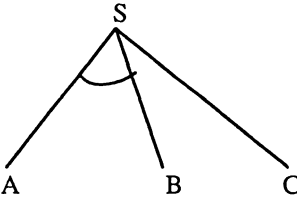
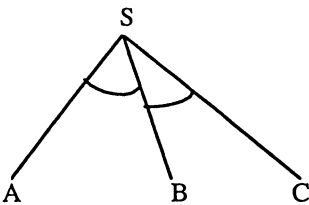
Example 10.1: Assume that the designer's knowledge body is in the form of production rules (the score values will be used later) as in Table 10.2, and each production rule is a conjunction of several attributes. Suppose that the initial specification is r_1 . In order to reach an accepting state, the designer may try to find a rule that decomposes r_1 ; i.e., a rule that has r_1 as a conclusion (on the right-hand side). The only candidate rules are 3 and 6, but 6 is chosen since it has higher score value. For the new process step the designer establishes a new sub-goal to decompose r_2 . If r_2 can be matched with structural attributes, then r_1 could be satisfied by *modus ponens*. The designer's next sub-goal is to decompose r_3 and m_1 ($r_3 \wedge m_1 \Rightarrow r_2$). Since m_1 is a structural attribute, it is only necessary to decompose r_3 . Thus the process proceeds as shown in Table 10.3. Note that if the designer had chosen to decompose r_1 by applying rule 3, a different sequence of production rules would result. Figure 10.1 illustrates the AND/OR search process for the whole tree. As shown in Figure 10.1, the r_1 specification is satisfied by the following accepting process states:

- | | |
|---|---------------------|
| 1. $m_1 \wedge m_2 \wedge m_3 \wedge m_4$ | 4. $m_1 \wedge m_6$ |
| 2. $m_1 \wedge m_4 \wedge m_2$ | 5. $m_1 \wedge m_5$ |
| 3. $m_1 \wedge m_4 \wedge m_7$ | |

These process states represent alternative design solutions; each derived from a different sequence of production rules. Such solutions are represented in the solution space as conjunctions of structural attributes. For example, the derivation of solution 1 was determined by the sequence of production rules as shown in Table 10.3. In Figure 10.1, there are five different specification levels for abstraction and

complexity, and each level represents a design problem with different degree of detail. The first one is represented with specifications r_2 and r_8 . The second level is represented with specifications r_3 and r_9 or r_{10} . The third level is represented with specifications r_4 and r_5 or r_{11} . The fourth level of abstraction is represented with the specification r_6 .

Table 10.1 Graphical Representation of AND/OR clauses

Graphical Representation	Interpretation
	$\underline{A \vee B \Rightarrow S}$ <p>S is satisfied if A or B are satisfied</p>
	$\underline{A \wedge B \Rightarrow S}$ <p>S is satisfied if A and B are satisfied</p>
	$\underline{(A \wedge B) \vee C \Rightarrow S}$ <p>S is satisfied if (A and B) or C are satisfied</p>
	$\underline{(A \wedge B) \vee (B \wedge C) \Rightarrow S}$ <p>S is satisfied if (A and B) or (B and C) are satisfied</p>

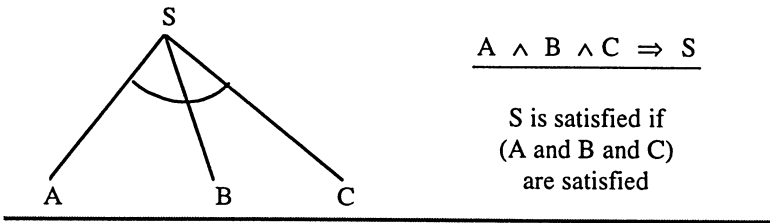


Table 10.2 Production Rules for Example 10.1

				Score
Rule 1:	$m_2 \wedge r_4 \wedge r_5$	\Rightarrow	r_3	2
Rule 2:	$r_3 \wedge m_1$	\Rightarrow	r_2	2
Rule 3:	$m_1 \wedge r_8$	\Rightarrow	r_1	1
Rule 4:	$m_4 \wedge r_9$	\Rightarrow	r_8	2
Rule 5:	r_6	\Rightarrow	r_5	2
Rule 6:	r_2	\Rightarrow	r_1	2
Rule 7:	r_{10}	\Rightarrow	r_8	1
Rule 8:	$m_1 \wedge r_{11}$	\Rightarrow	r_9	2
Rule 9:	m_3	\Rightarrow	r_4	2
Rule 10:	m_6	\Rightarrow	r_{10}	1
Rule 11:	m_5	\Rightarrow	r_{10}	2
Rule 12:	m_2	\Rightarrow	r_{11}	2
Rule 13:	m_7	\Rightarrow	r_{11}	1
Rule 14:	m_4	\Rightarrow	r_6	2

Table 10.3 Illustration of the Process History for Example 10.1

PROCESS STEP	PROCESS STATE	PRODUCTION RULE
1	r_1	6
2	r_2	2
3	$r_3 \wedge m_1$	1
4	$m_1 \wedge m_2 \wedge r_4 \wedge r_5$	9
5	$m_1 \wedge m_2 \wedge m_3 \wedge r_5$	5
6	$m_1 \wedge m_2 \wedge m_3 \wedge r_6$	14
7	$m_1 \wedge m_2 \wedge m_3 \wedge m_4$	STOP

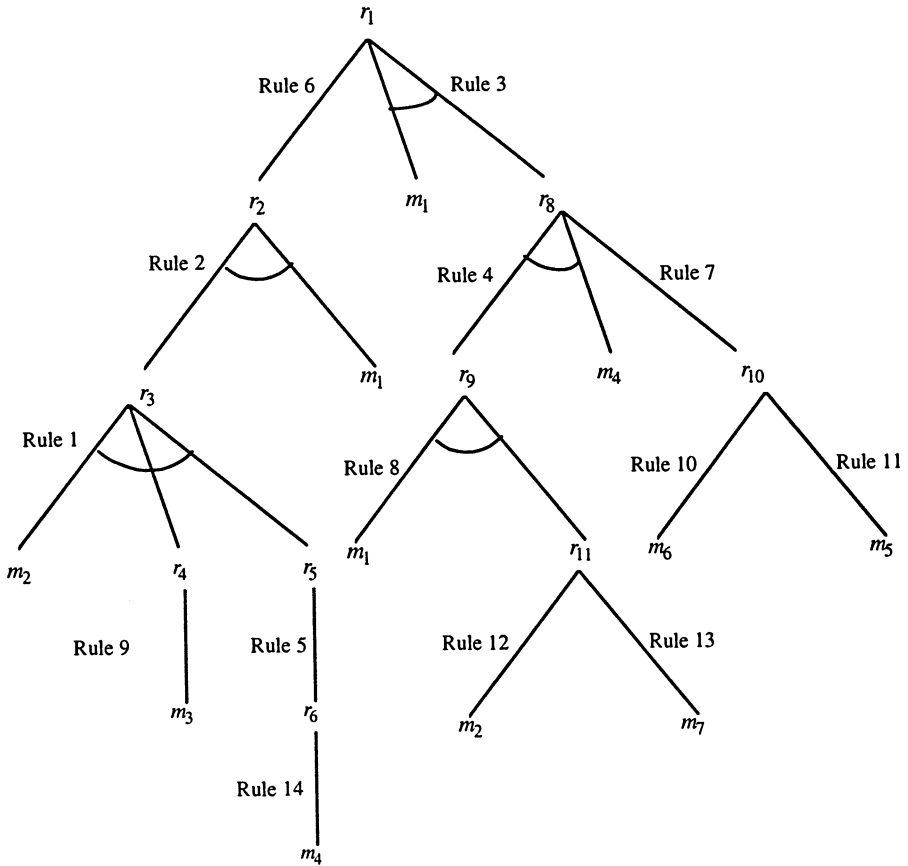


Figure 10.1 The Whole AND/OR Search Tree - Example 10.1

10.3 GUIDING THE AND/OR SEARCH TREE

As explained previously, a design search problem can be represented by searching through a branching AND/OR tree. The root of the tree represents the initial specifications to be satisfied, and the non-terminal nodes are either AND nodes or OR nodes. An AND node represents a specification or sub-specification that is satisfied only when all its children have been satisfied. An OR node represents a specification or sub-specification that is satisfied when any of its children has been satisfied. In Figure 10.1 $m_1, m_2, m_3, m_4, m_5, m_6,$ and m_7 are terminal nodes. r_1 is an OR node with two choices; rule 6 (node r_2), and rule 3 (the AND node m_1 and r_8). In Example 10.1, a backward-chaining system is used to decompose

specification r_1 . Rule 6 is tried first and when this path does not satisfy the initial specification, the system backtracks and rule 3 is tried.

The AND/OR tree for a complex system is typically large. In order to generate a solution, an extensive search of the state space might be required. This is mainly due to large volume of information and knowledge that exists in the form of production rules. In addition, specifications and production rules are often provided in an unstructured manner through a natural language interface. Thus, in many cases, the designer needs a systematic and analytic guideline to select alternative production rules (represented by an OR node) at each process state. There are a number of tree search techniques, which are divided into “blind search” and “heuristic search” [1] categories. The two basic methods of blind search are called “depth-first” and “breadth-first”. “Best-first” search methods are representative of heuristic searches. In this section, a search algorithm is introduced that combines the principles of best-first search methods and production rules.

The Design Search Algorithm

We now discuss a general type of design search algorithm based on the type-2 model introduced in Chapter 6 and the AND/OR tree representation. The idea is to select a specification at each step of the design process a specification that has been generated but not further investigated (‘expanded’), and to identify the most promising production rule that can decompose it. Each production rule is given a score by means of a *scoring* function that indicates the relative likelihood of being the right production rule. A production rule with the highest score value is selected. The algorithm makes use of a list of attributes, which are either a functional or structural, called OPEN or *candidate list*. The conjunction of all the attributes OPEN list represents the process state as illustrated in Table 10.3. The functional attributes in the OPEN list are active in the sense that they must be examined by the algorithm. The algorithm also makes use of a list of functional attributes, called CLOSE. Each functional attribute in the CLOSE list was already expanded by means of a *used* production rule. If a solution path ends with an inconsistent design, the OPEN and CLOSE lists are updated through the backtracking mechanism. The steps of the *Design Search Algorithm* are as follows:

Step 1: Initially, OPEN contains only the root node.

Step 2: If all the elements in OPEN are structural attributes and the description is consistent (i.e., represent a physically realizable design), terminate; if the description is inconsistent, go to Step 5; otherwise, go to Step 3.

Step 3: Expand the leftmost unexpanded functional attribute in OPEN (provided it is not already in CLOSE) to generate its child attributes. Remove it from the queue OPEN, and place it at the beginning of the queue CLOSE. If only one production rule is generated then add the child attributes at the beginning of the queue OPEN and go

to Step 2; otherwise, go to Step 4.

Step 4: If several production rules are generated, then the *unused* production rule with the highest score value is selected. Add the child nodes at the beginning of the queue OPEN, and go to Step 2.

Step 5: Identify the leftmost attribute in CLOSE, which can be expanded with unused production rules. Place the attribute at the beginning of the queue OPEN, and remove its descendant attributes in OPEN and CLOSE. If all the attributes in CLOSE can not be further expanded, terminate; otherwise, go to Step 3.

To clarify the *Design Search Algorithm* consider the following example.

Example 10.2: Assume that the production rules and their score values are those expressed in Table 10.2. Suppose that the initial specification is r_1 , and that both structural attributes m_1 and m_2 can not be included in a physically realizable design. The whole *Design Search Algorithm* proceeds as shown in Table 10.4.

Table 10.4 The *Design Search Algorithm* applied to the AND/OR tree of Figure 10.1

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	r_1	\emptyset	6, 3	6
2	r_2	r_1	2	2
3	r_3, m_1	r_2, r_1	1	1
4	m_2, r_4, r_5, m_1	r_3, r_2, r_1	9	9
5	m_3, m_2, r_5, m_1	r_4, r_3, r_2, r_1	5	5
6	r_6, m_3, m_2, m_1	r_5, r_4, r_3, r_2, r_1	14	14
7	m_4, m_3, m_2, m_1 (inconsistent solution)	$r_6, r_5, r_4, r_3, r_2, r_1$	backtracking	--
8	r_1	\emptyset	3	3
9	m_1, r_8	r_1	4, 7	4
10	m_1, r_9, m_4	r_8, r_1	8	8
11	m_1, r_{11}, m_4	r_9, r_8, r_1	12, 13	12
12	m_2, m_1, m_4 (inconsistent solution)	r_{11}, r_9, r_8, r_1	backtracking	--

13	r_{11}, m_1, m_4	r_9, r_8, r_1	13	13
14	m_7, m_1, m_4	r_{11}, r_9, r_8, r_1	STOP	--

10.4 A PROTOTYPE SYSTEM TO IMPLEMENT THE DESIGN SEARCH ALGORITHM

The decomposition of specifications performed by Step 3 and Step 4 of the Design Search Algorithm can be done by a designer, an intelligent advisor, or both. A designer performs decomposition according to his/her experience and design knowledge. An intelligent advisor is a knowledge-based system, which is supported by catalogues, containing the principles (i.e., production rules) of specification decomposition.

In this section, we give an illustration of the practical usefulness of our framework. We discuss a prototype system called CADAT (CAse-based Design Advisory Tool), which has been implemented on an IBM PC using Case-1™ [2]. Case-1 has rapid memory access to a large body of knowledge that contains a set of general production rules, also termed as *cases*. The cases, which have been developed from previously designed artifacts, are stored in a computer database. They are used to guide the design process in decomposing specifications into sub-specifications and/or the corresponding structural properties. Cases can be retrieved from the memory using several indexing techniques that make searching the knowledge base more efficient and effective. Case-1 also provides easy-to-use tools for adding to that body of knowledge.

10.4.1 CASE-1 OVERVIEW

Case-1 [2] involves up to three modules that allow for user interaction: case builder, case retrieval and synthesis, and system administration. The components of Case-1 that do not require user interaction include: the reasoning engine, and the database. These modules are summarized as follows:

- *The Case Builder* - the Case Builder is the authoring tool used by knowledge engineers to create and modify cases (production rules). *Cases* are the atoms of the Case-1 system. They may be thought of as a single potential decomposition of a specification into sub-specifications, or the corresponding structural attributes. A case has information associated with it that allows Case-1 to determine if the case is relevant to specifications ("problem description") the designer might present. This information is comprised of the textual description of the specification and decomposition ("solution"), a list of qualifying questions, and any appropriate hypermedia attachments. All of these components are used by the Analyzer to present

™ Case-1 is a trademark of ASTEA International, 1995.

to the designer the most probable set of decompositions for the specification.

- *The Analyzer* - this module allows the user to present problems (specifications) and obtain solutions (through decomposition). Each new problem entered into the Case-1 analyzer is called a *session*. Designers compose sessions interactively, with the help of editing tools including high level spell checker. The designer then initiates a search of the Case-1 database. When the search is complete, Case-1 displays a sorted list of the most probable answers to the design problem. If the solution list is not satisfactory, then the Analyzer allows for search criteria to be changed by either narrowing or broadening the search area. Interactive questions and answers allow for further refinement of the solution set. If no satisfactory solution is presented by Case-1, the session can be placed in a special queue for future review by a knowledge engineer. Once the session is complete, it can be closed and saved for future operation.

The list of used sessions assists the designer in representing and recording the list OPEN as defined by the *Design Search algorithm* (see Section 10.3). If all the elements in OPEN are structural attributes that represent a physically realizable design, the search is terminated. Otherwise, a new session is started where the Problem Description window of the new session includes the description of the leftmost unexpanded functional attribute in OPEN.

- *The System Administration Utility* - this module allows the designer to select options and to fine-tune the operation of Case-1. The success of a knowledge-based tool such as this depends solely on the data it contains.
- *The Reasoning Engine* - this is where all intelligence within Case-1 resides. It communicates with the database and also interfaces with the interactive components.
- *The Database* - the Case-1 database contains all the information that Case-1 needs in order to operate including the cases, case statistics, session status, and attachment file locations.

The CADAT architecture is summarized in Figure 10.2. The “SYNTHESIS” and “DESIGN KNOWLEDGE” components are currently the prime focal points of a Case-1 implementation. The “EVALUATION” component of CADAT is concerned with assessing the validity and critiquing the design solutions relative to the original functional requirements (which is currently not part of the Case-1 implementation). Since the knowledge base is generally only heuristically suggestive of causal design relations, it can be improved or repaired by processing the functional performance evaluation of the artifact. The performance evaluation could be provided by means of a simulation (e.g., finite element analysis). In many cases this simulation could be provided by a human designer, since designers can usually envision artifact performance in many domains.

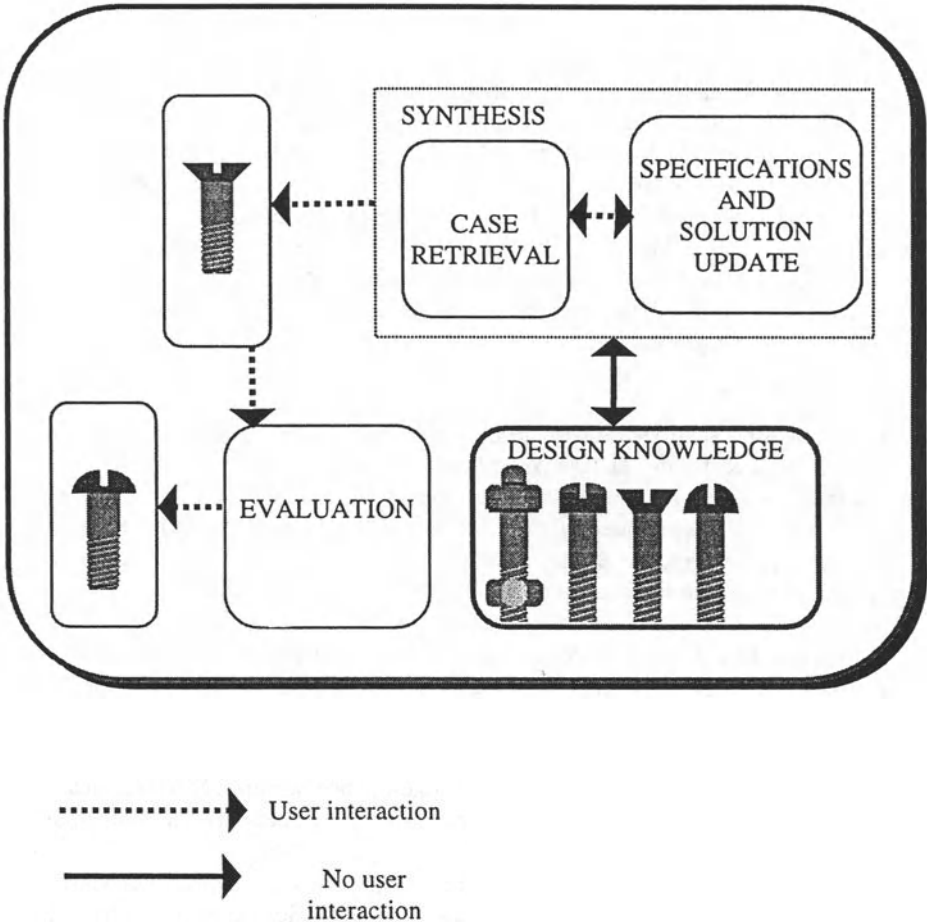


Figure 10.2 The CADAT Architecture

10.4.2 BASIC CASE-1 DEFINITIONS

Case

Each production rule is considered a case. By grouping cases into classes, a domain can be created for a set of production rules that pertain to a particular area of knowledge (e.g., fasteners, cars, forklifts). A case consists primarily of the problem description (representing the specification that needs to be expanded) and the solution (representing the child attributes). Here are some simple case examples (UPPER CASE characters represent functional attributes while lower case letters denote structural attributes):

Table 10.5 Examples of Cases (Representing Production Rules)

Problem Description	The car is ECONOMICAL
Solution	The car has RELIABLE TIRES & LOW FUEL CONSUMPTION
Problem Description	The car is SAFE FOR HIGH SPEED DRIVING
Solution	The car has 4-wheel steering (4ws) & stabilizers in the front suspension systems & GOOD BRAKES & extra strong roof & rigid passenger compartment

Each case has its associated information decomposed and structured so that Case-1 can determine the relevance of each case to specifications the designer might present. When a designer enters a problem description into the Analyzer and asks for solutions, Case-1 presents a list of cases, ranked by the reasoning engine's estimate of each case's quality and likelihood of solving the problem.

In addition to the descriptive features, each case can have an associated list of questions and potential answers. Depending on the answers that the designer provides, Case-1 may give that case a higher or lower rank.

Each case may also have one or more hypermedia attachments. Information in the attachments is not considered by the Reasoning Engine, but can provide additional information to the designer.

The scoring of cases with respect to a particular problem statement is affected by a number of parameters. The parameters are dynamically adjusted by the system to reflect actual operational experience as the system is run.

Sessions

A session is an individual instance of a problem statement (i.e., specification description) presented to the Analyzer. For instance, a problem statement could be:

"The fastener is EASILY DISASSEMBLED"

"The computer classroom has HIGH LEVEL ACOUSTICS"

In order to gain the best set of potential answers from Case-1 the problem description must be as complete as possible.

Domains

A domain is a set of cases that pertain to a particular area of design knowledge. The

designer selects the domains to be searched during each session. Only solutions in the selected domains will be presented to the designer. The designer can change the domain selections interactively during a session.

Questions

During the typical Case-1 problem resolution session (specification decomposition), a set of questions are presented to the designer. Depending on the answers to the questions, a different list of proposed production rules (cases) may be presented to the designer.

The answers designers give to questions are one of the key sources of information that Case-1 uses to rank production rules. The designer may initially choose to answer as many questions as seem appropriate. Additional questions can be answered interactively later in the process to further refine the ranking of the production rule list.

The Analyzer Interface

The designer poses design problems to Case-1 through the Analyzer. This is done by starting a new “session,” and including a description of the specification in plain English. When the description is complete, the Case-1 Reasoning Engine uses artificial intelligence techniques to sift through its knowledge base, and provides a list of possible production rules or cases. If no exact match is available, Case-1 presents the closest solutions ranked by their likelihood of solving the problem.

In order to refine the search, Case-1 also poses a list of questions, each with multiple choice answers. The designer can respond to some or all of the questions by selecting answers from each question’s pull-down list. The Case-1 Reasoning Engine uses the answers to update the production rule selection and to present a new list to the designer. The process is interactive and can be repeated as needed to find the right decomposition.

Questions, answers and production rules (cases) can have remarks or notes attached to them that designers can view as needed. The notes may include hypermedia documents (such as drawings).

The Reasoning Engine

Case-1 uses several indexing techniques to make searching the knowledge base more efficient and effective. These include: common word elimination, morphological pre-processing, synonym consolidation, tolerant pattern matching, and index matching. Case-1 takes the specification description entered by the designer and compares it word by word to the production rule database. In order to find the most likely production rule to a specification, Case-1 searches its knowledge base for similar

production rules and, using intelligent statistical manipulation, displays a list of the most probable solutions. Thus, the more often elements appear within a production rule, the more likely that production rule will contain the most appropriate solution.

Case-Builder Interface

Cases are written by knowledge engineers who have expertise in the problem area. When developing cases for a new problem domain, the knowledge engineer typically outlines the problem area and identifies significant and relevant functional and structural attributes. The knowledge engineer then breaks this information down into a series of cases; each dealing with one specific production rule. The knowledge engineer then develops a case title, problem description and solution description. The knowledge engineer then assigns questions and answers to each case. The engineer also assigns weight to the answers based on his/her evaluation of each answer's alignment with the case, and likelihood of leading to the best decomposition.

10.4.3 THE CASE BUILDER INTERFACE

Case Builder is the authoring tool used by knowledge engineers to create and modify production rules. When creating or editing a rule, the engineer must enter information that will allow the reasoning engine to determine if the production rule is relevant to specifications the designer might present. The knowledge engineer also associates a list of questions and answers to the production rule. Finally, the knowledge engineer can supplement the information available to the designer by adding hypermedia (such as drawings) and other attachments.

Specifying A Domain

Since production rules are organized into knowledge domains, we must first specify the domain in which we want to work. We select Open from the Case Builder File menu (or the open folder icon from the toolbar). The Domain dialog box appears and displays a list of available domains (see Figure 10.3). We may then select the name of the desired domain and then click OK.

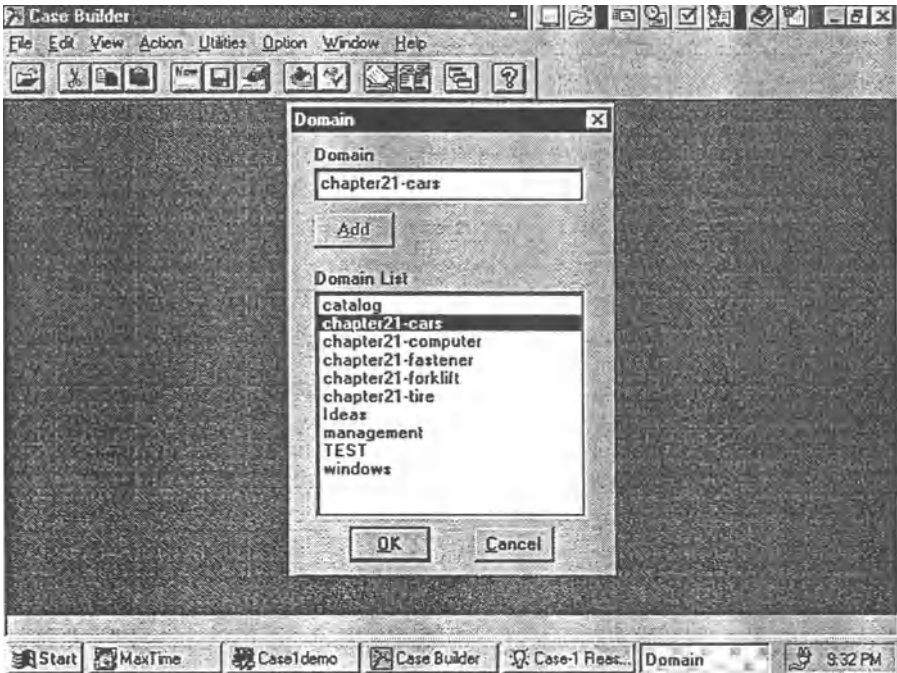


Figure 10.3 The Domain Dialog Box

If we want to create a new domain, we type the name we want to give the new domain into the Domain text field. When we click OK, we should see the main Case-Builder window. At the top are three tabs labeled Cases, Details and Questions (see Figure 10.4). The leftmost tab, Cases, shows a list of all production rules contained in the domain.

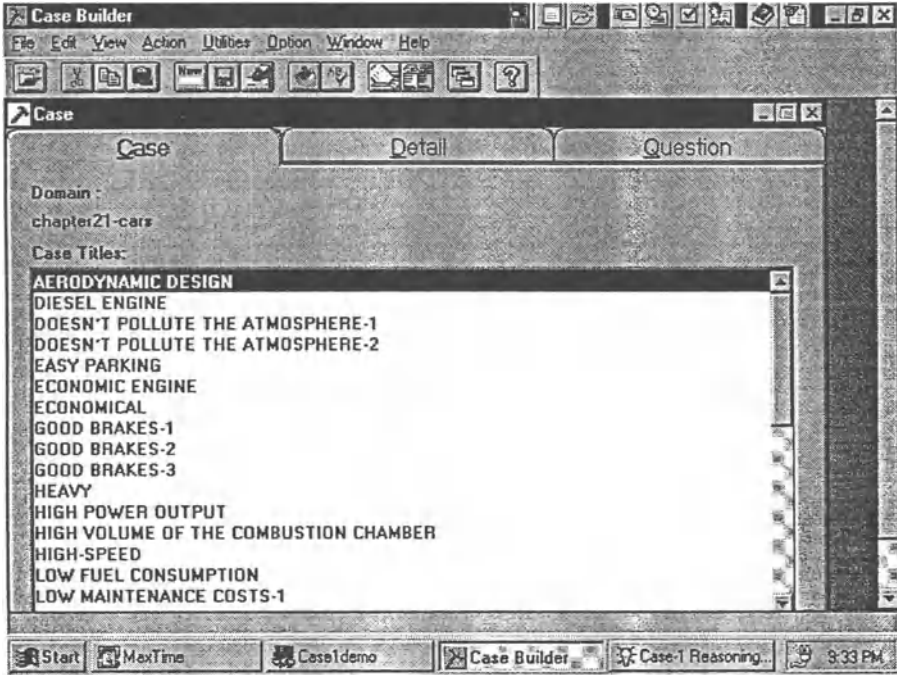


Figure 10.4 The Main Case Builder Window

Creating A New Case

We create a new case when we need to add a production rule that does not already appear in the knowledge base. To start building a new case, we select New from the Case Builder menu. The Detail tab will come to the front as shown in Figure 10.5. We need to enter a title for the case in the Title text field. This title is the text that appears in the list of cases during a session search. We should also enter a complete description of the specification in the Case Problem Description text field. The information in this field is what the reasoning engine uses to match against session problem descriptions. The Solution Text field should contain a series of sub-specifications. Thus, to determine the Case Problem Description, the sub-specifications should be developed first. When we are satisfied with the case, we add it to the domain database by selecting Save from the action menu.

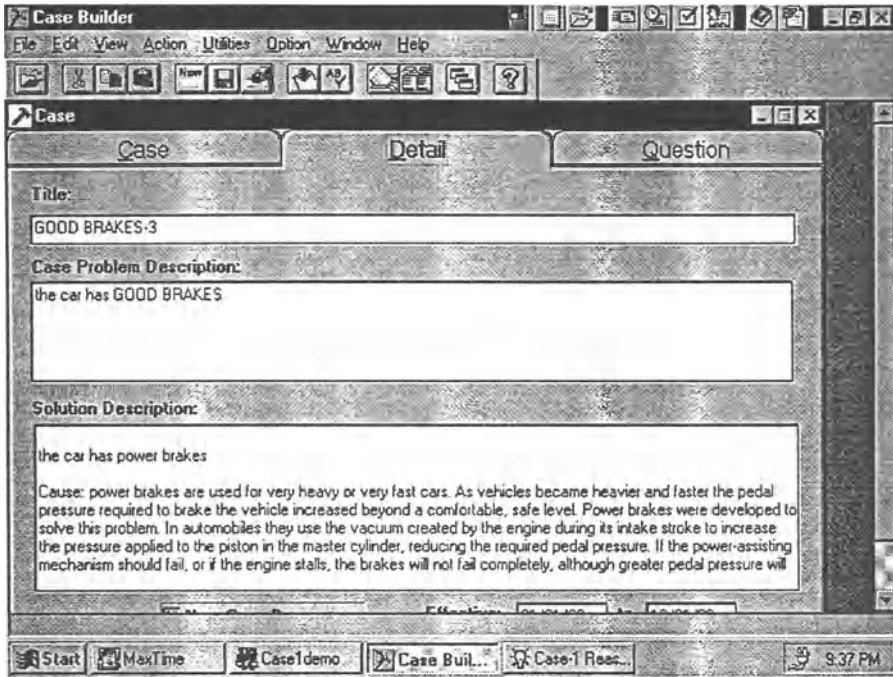


Figure 10.5 Entering A New Description of A Production Rule

Attaching Questions to A Case

When Case-1 analyzes a problem statement, a set of discriminating questions is usually presented to the designer. These questions help the Analyzer to narrow the search, and the Search Engine ranks production rules differently depending on the answers to the questions. As a result, a different order of production rules may be presented to the Analyzer operator.

The answers designers provide to the questions are key sources of information used by Case-1 to rank production rules. The designer may initially choose to answer as many of the questions presented as seems appropriate. Additional questions can be answered interactively to further refine the candidate list of production rules later in the process.

The set of questions is common to all the production rules in a domain. The answers and their scoring weights are part of each production rule. The knowledge engineer working on a production rule may choose to assign answers and weights to only part of the questions presented in the case's domain.

To activate the Case Question Screen, we click on the Question tab (see Figure 10.6). To add a question, we click on the Question tab and then select New from the Case Builder Action menu. We can then type the question into the Question text field. We may assign a weight to a question as it applies to the production rule we are

working on. The weight ranges from 0 to 1, reflecting the importance of the Question. A higher weight indicates a more important question.

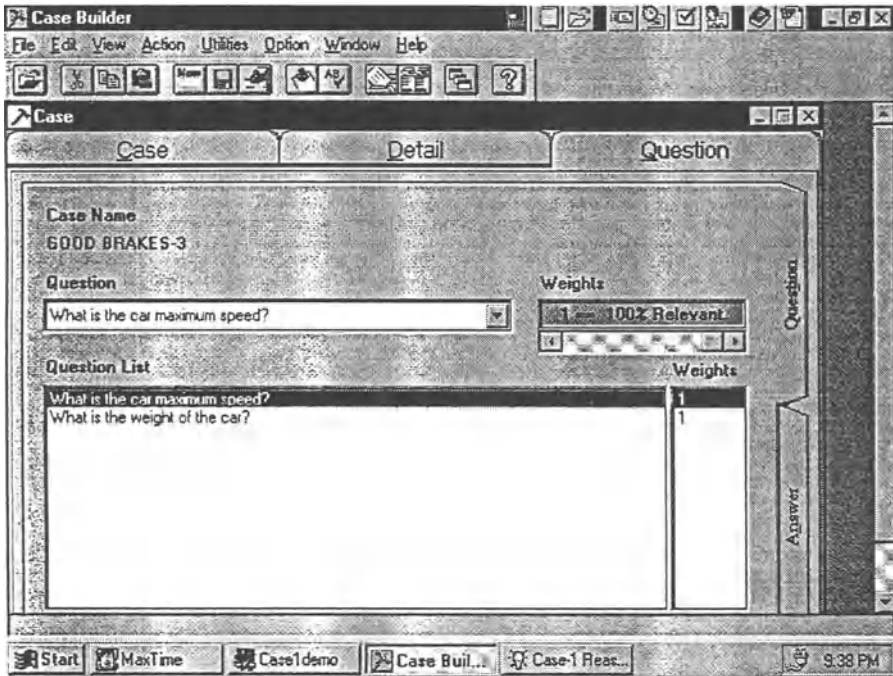


Figure 10.6 Activating the Production Rule Questions Screen

Attaching Answers to Questions

For each question, Case-1 proposes all available answers. The answers selected by the designer affect how the production rules are scored, and influence which production rules are presented and in what order.

To activate the Answer screen, we click on the Answer tab as shown in Figure 10.7. To create a new answer, we select New from the Case Builder Action menu, type the answer in the Answer text field, give the answer an appropriate weight for the production rule, and select Add from the Case Builder Action menu.

An answer weight is a number between -1 and $+1$. When the designer selects an answer, the Reasoning Engine reviews the weights assigned to that answer in each of the production rules. If the weight is positive, the Reasoning Engine assigns a higher likelihood that the production rule is the correct one. If the weight is negative, the Reasoning Engine assigns a lower likelihood. If the weight is zero, the question is considered neutral and the likelihood is not changed. For example, suppose we are working on the production rule presented in Figure 10.5 (“IF the car has power brakes THEN the car has GOOD BRAKES”). We add the question that asks: What

is the car's maximum speed? As shown in Figure 10.7, the possible answers that are created are:

- 100 Km/h;
- 160 Km/h;
- over 210 Km/h

Since power brakes are used for very heavy or very fast cars, we might assign a weight of 1 to “over 210 Km/h,” a neutral score of 0 to “160 Km/h,” and a weight of -1 to “100 Km/h.” Thus, we assigned a weight of -1 to “100 Km/h” since it is unlikely that power brakes are needed on such a slow vehicle (drum brakes can be used instead).

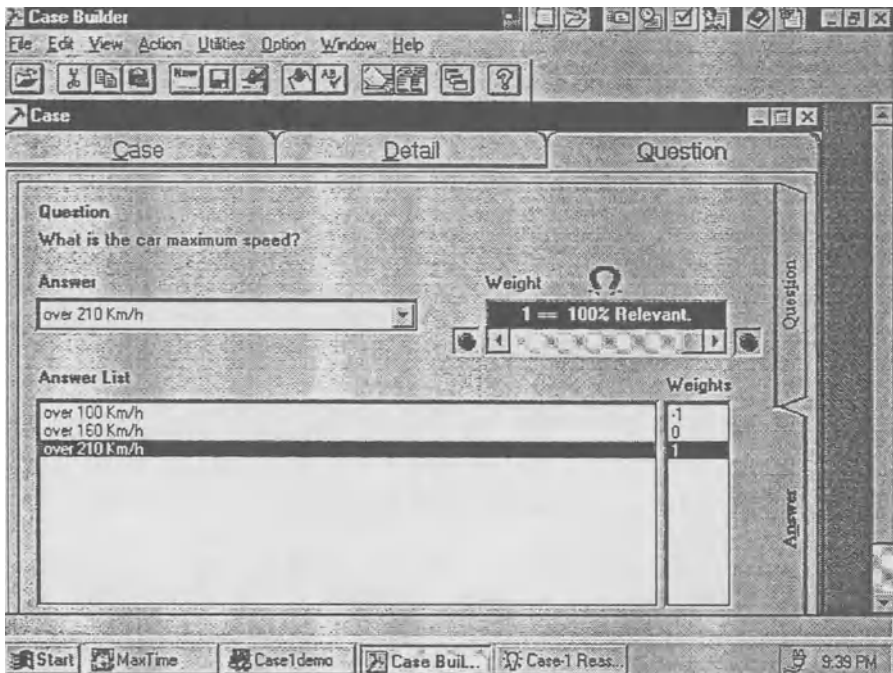


Figure 10.7 Attaching Answers to Questions

Attaching Notes

Attachments are informative notes (e.g., technical drawings, images, video clips) that can be added to Case-1 sessions, cases, questions, and solutions. The designer is presented with a list of available attachments and may choose to view them or not. The information in the attachment is for the designer's benefit only and is not analyzed by the Reasoning Engine. For example, a media clip describing the

operation of a hydraulic disk brake might be attached to a case that says “IF the car has hydraulic disk brakes, THEN the car has GOOD BRAKES”. Thus, more clearly illustrating the component parts and operating mode (see Figure 10.9).

To activate the Attachment Notes window, the designer selects Attachment Info from the Case Builder Action menu. The Attachment Notes window will appear as shown in Figure 10.8. To add a new Note, the designer selects the OLE type (a Microsoft standard for exchanging information between different Windows applications) of the file containing the Note, and indicates which file contains the Note information. The knowledge engineer may also want to test the information in the selected source document file by clicking on the Test button (see Figure 10.9).

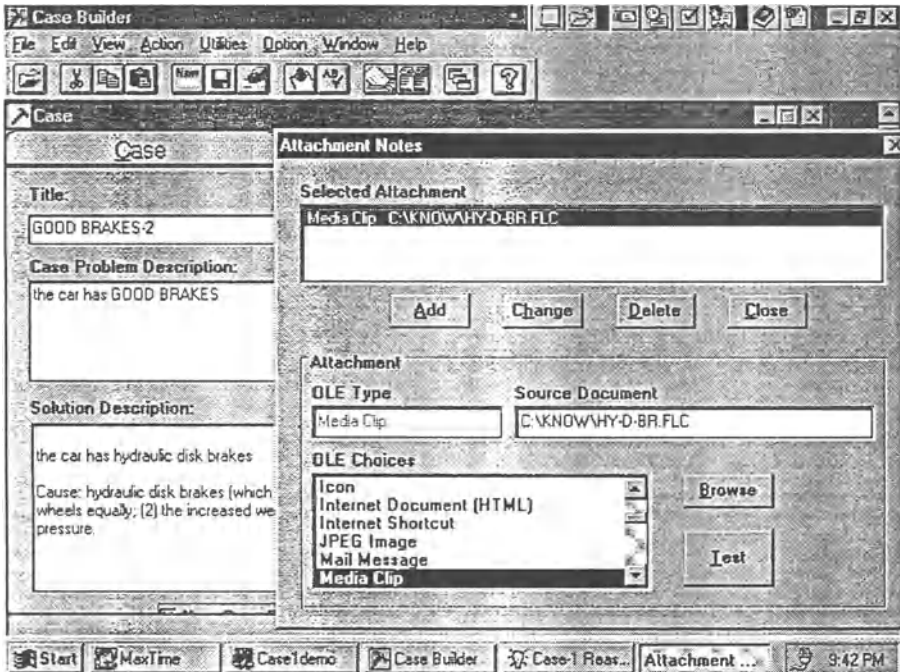


Figure 10.8 Attaching Informative Notes to A Production Rule



Figure 10.9 Testing the Informative Note Related to Hydraulic Disk Brakes by Clicking on the Test Button

10.4.4 THE ANALYZER INTERFACE

The Analyzer is the designer's primary access to Case-1 implementation of the Design Search Algorithm presented in Section 10.3. It allows the designer to present a specification and obtain a list of sub-specifications in a highly intuitive, natural manner. Each new specification description is called a *session*. When the designer is done describing the specification, a search of the production rule database begins. When complete, Case-1 displays a sorted list of cases that represent the most relevant set of potential production rules in the knowledge base.

If Case-1 presents a production rule that is satisfactory to the designer, the list of attributes in OPEN is updated as described by Steps 3 and 4 of the Design Search Algorithm, and the session is closed. The maintenance of the list OPEN, which includes the "active" functional attributes as well as the partial design solution, is not implemented by Case-1. The list OPEN may be maintained manually, or by using any standard online Windows editing tool (e.g., Notepad). If no satisfactory production rule is presented, the session can be placed on a special queue for review by a knowledge engineer.

Starting a New Session

Assume that the designer is faced with the problem of designing a car that is able to achieve the following specifications (see Section 21.1 for automobile design example):

1. The car is SAFE (r_1);
2. The car is capable of HIGH SPEED (r_2);
3. The car has LOW FUEL CONSUMPTION (r_3).

The initial list of attributes in OPEN includes $\{r_1, r_2, r_3\}$. We can start a new session by selecting New Session from the Case-1 Analyzer edit menu. When we have completed naming the session and selecting the car design domain to match our problem (see Figure 10.10), we click OK. The Session window appears as shown in Figure 10.11. Within this window there are three screens represented by index tabs: Session, Examine, and Question. The Session tab allows the designer to compose a specification description, the Examine tab enables the designer to look at a production rule in more detail, and the Question tab lets the designer answer questions that help refine the search.

Since many production rules may match the current list of attributes in OPEN, the preferred rule is the one that matches the first leftmost of the specifications in the list OPEN. Thus, the designer composes a problem by describing the specification “The car is SAFE” in the Problem Description window of the Session tab, as shown in Figure 10.11.

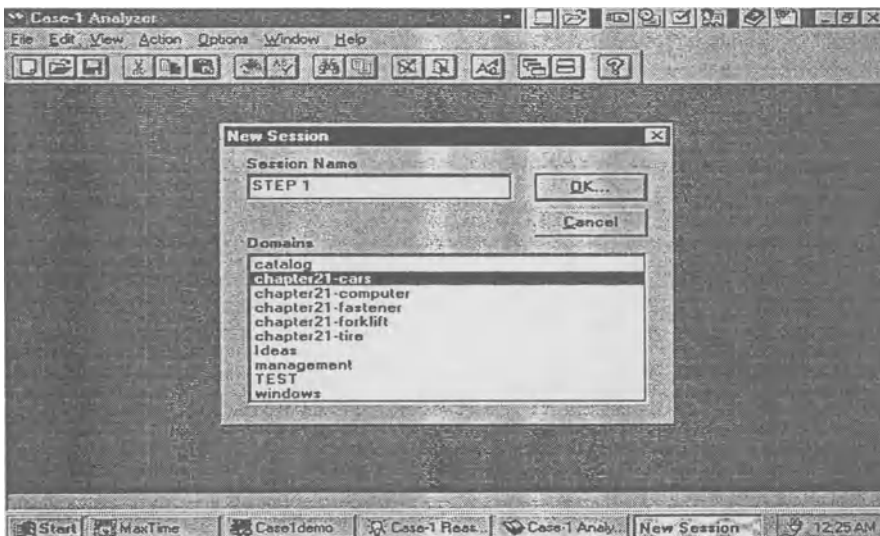


Figure 10.10 Starting A New Session by Selecting the Appropriate Domain

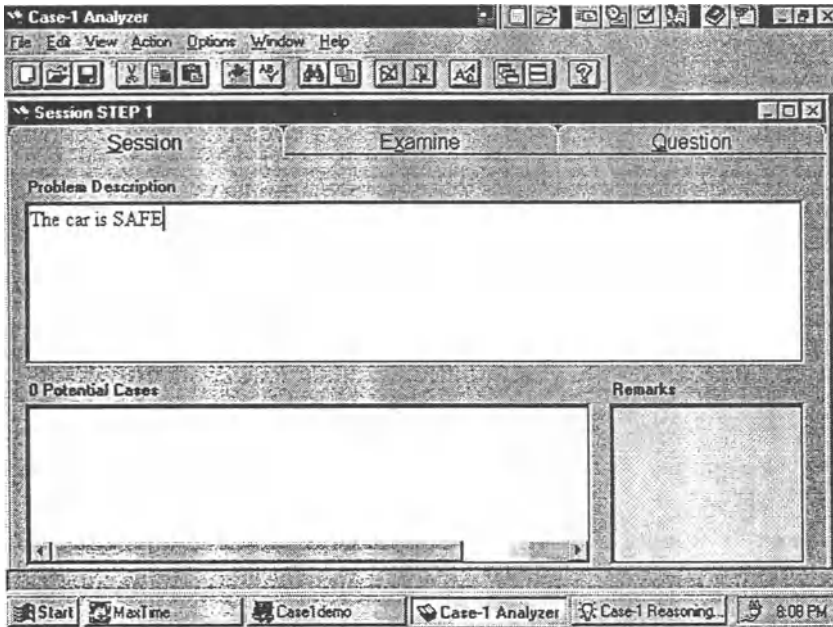


Figure 10.11 Composing A Problem by Describing the Specification in the Problem Description Window of the Session Tab

Launching the Search

Once we have entered the specification description “the car is SAFE,” we start the search by selecting Search from the Action menu. When the search process is complete, a ranked list of potential production rules is displayed as shown in Figure 10.12. The score next to each production rule indicates its relative likelihood of being the right decomposition for the particular specification. The production rules are displayed in order of rank (score).

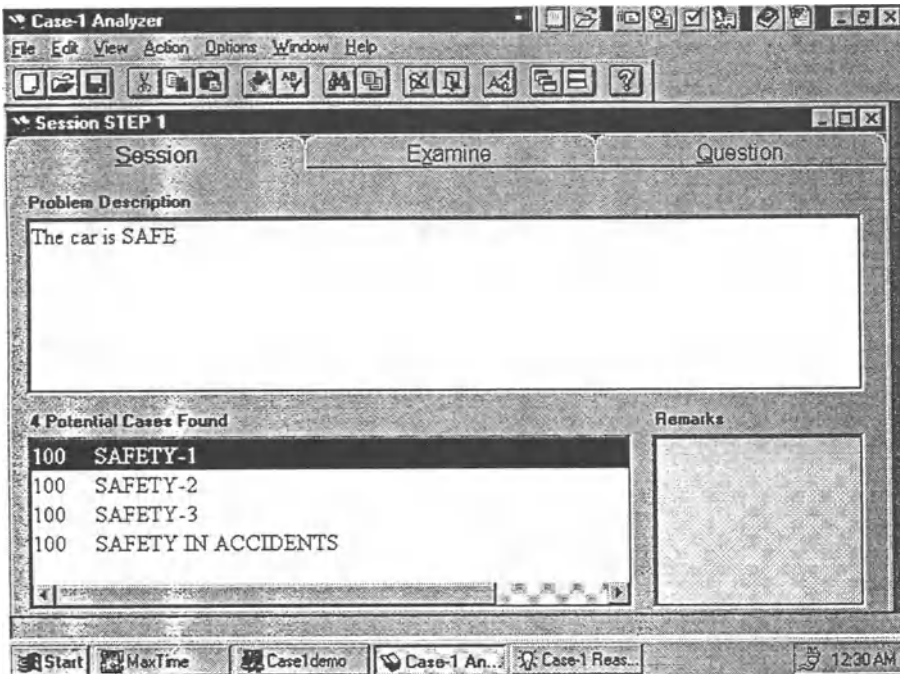


Figure 10.12 A Ranked List of Production Rules is Displayed When the Search Process is Complete

Answering Questions

If we want to narrow down the set of production rules, we click on the Questions tab and respond to questions stored with the production rules. We select questions by clicking on a question in the Question window as shown in Figure 10.13. We answer questions by clicking on the desired answer in the Answer window. In our example, we answer the questions (see Figure 10.13) according to the following assumed constraints: (1) the external conditions are good, (2) the maximum speed is 160 Km/h, and (3) the weight of the car is between 1 and 3 ton. Once we have answered the pertinent questions, we restart the search by clicking on the Session tab. As shown in Figure 10.14, the reasoning engine uses the questions to update its production rule selection and presents a new list to the designer.

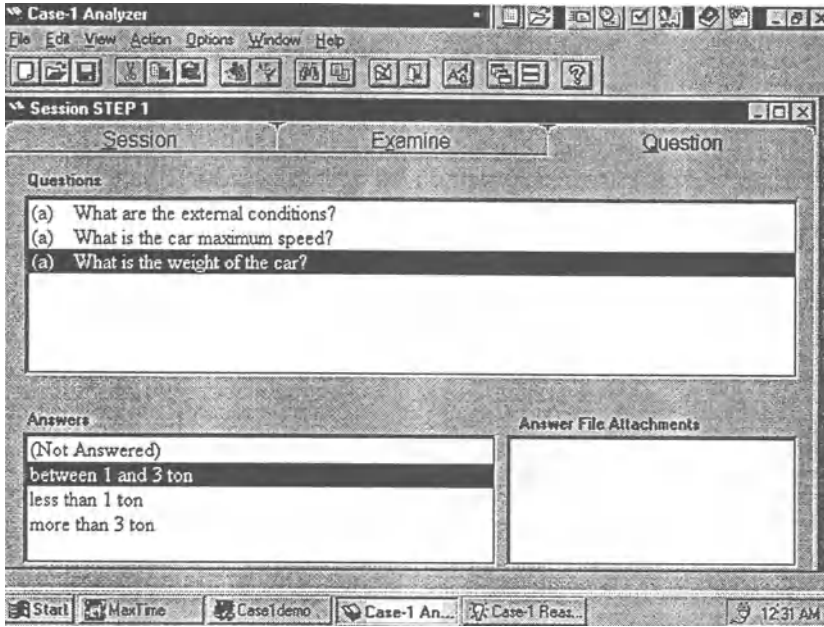


Figure 10.13 Selecting the Questions and Answers

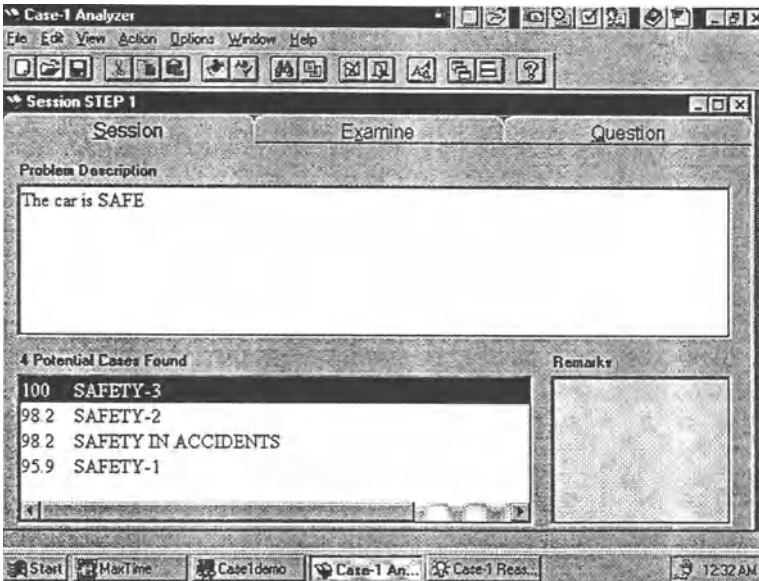


Figure 10.14 The Answers to Questions are Key Sources of Information used to Rank Production Rules

Examining and Selecting a Preferred Production Rule

During a session, we want to review the candidate set of production rules and decide which seems the most likely decomposition of the particular specification. For example, to see more information about the production rule “SAFETY-3,” we select it and click the Examine tab. The Examine screen shown in Figure 10.15 appears.

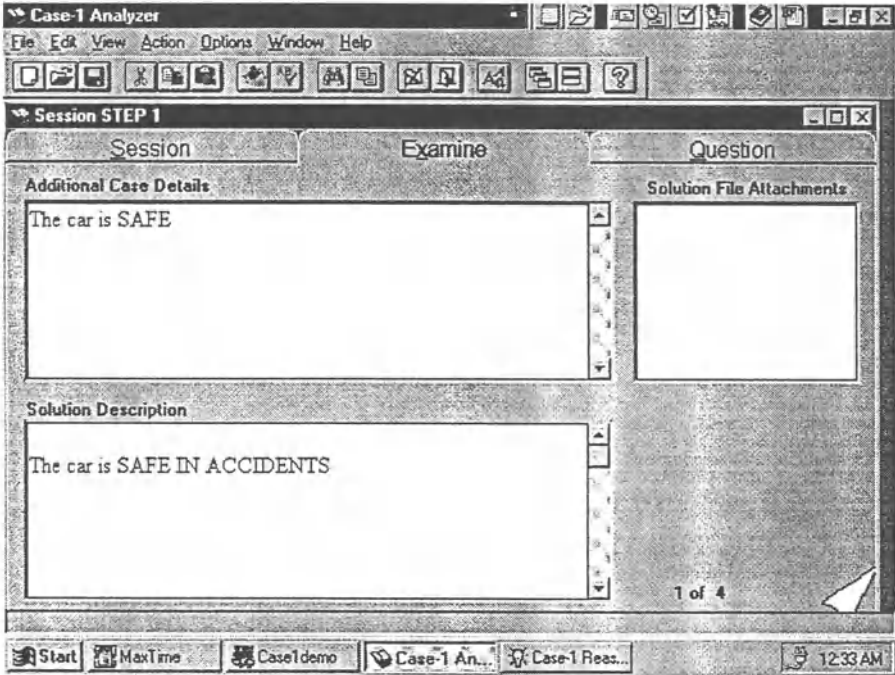


Figure 10.15 Examining A Potential Production Rule

The Examine screen has three text fields. The first gives more information on the specification that the production rule is decomposing. The second gives the list of sub-specifications, and the third shows a list of hypermedia attachments that further illustrate the specification or its list of sub-specifications.

As mentioned earlier, the list OPEN may be maintained by using any standard online Windows editing tool. For example, we can select the list of sub-specifications from an Examine screen text box and then select Copy from the Edit menu to copy that text to the Windows clipboard. We can then paste that information in any Windows application that maintains the list OPEN.

In our example, we may decide that the preferred rule is “SAFETY-3” (ranked first). We then remove the specification “the car is SAFE” from the queue OPEN, and add the specification’s sub-attribute “the car is SAFE IN ACCIDENTS” (r_4) at the beginning of the queue OPEN. The current session is closed, and the search

process is then applied (by starting a new session) recursively on the sub-specifications in the updated list OPEN (i.e., $\{r_4, r_2, r_3\}$). The process continues until eventually a set of structural attributes that correspond to the sub-specifications is identified. Table 1 of Chapter 20 shows all the process states generated in the course of searching for a solution to the automobile design problem.

10.5 SUMMARY

In summary, we introduced the Design Search Algorithm that is based on a set of production rules and is used to search for a physically realizable design solution. The main idea of the Design Search Algorithm is to select, at each step of the design process, a specification that has been generated but not investigated ('expanded'); and to identify the most promising production rule that can decompose it. Each production rule is given a score (by means of a *scoring* function) that indicates its relative likelihood of being the best production rule for the specification. Thus, a production rule with the highest score value should be selected.

The Design Search Algorithm has been implemented on a prototype system called CADAT (CASe-based Design Advisory Tool). The CADAT system performs production rule retrievals, which are stored in the production rule memory. To do this, the designer starts a new session and types a new specification (the leftmost unexpanded specification in OPEN) in plain English. A production rule has information associated with it that allows the reasoning engine to determine if the production rule is relevant to specifications the designer might present. This information is comprised of the textual description of the production rule, a list of qualifying questions and answers, and finally any appropriate hypermedia attachments. The reasoning engine, via its knowledge of the existing production rules and the answers that the designer entered, presents a list of possible production rules using several indexing techniques (based on the relevance of the production rule and common words). The designer reviews the unsatisfied specifications to determine which physical components satisfy the initial design specifications. This results in a solution alternative.

The Design Search Algorithm with its CADAT implementation has the following characteristics: (1) it provides a systematic approach to guide the search in a large and complex solution space; (2) it uses a set of production rules based on previous "good" designs to guide the search; (3) each production rule has its associated information decomposed and structured so that its score can be determined; (4) the production rules in the knowledge base are not only made within the context of the task in question, but also imply a variety of tacit assumptions (e.g., environmental conditions). CADAT uses a set of questions and answers to better rank the production rules, and to enable production rules to be applied to other contexts.

Finally, the conceptual design procedure we have described will not improve in performance over time. This is because there is no feedback mechanism, or evaluation of a design result, which enables the system to learn new causal relations with experience. In order to resolve this issue, tools (e.g., simulators) that provide

performance evaluations should be developed in the future.

REFERENCES

1. Luger, G. F. and Stubblefield, W. A., *Artificial Intelligence and the Design of Expert Systems*. Redwood City, CA: Benjamin/Cummings, 1989.
2. Case-1 User Manual, ASTEA International, 1995.

CHAPTER 11

PHYSICAL DESIGN OF PRINTED CIRCUIT BOARDS: GROUP TECHNOLOGY APPROACH

In this chapter, the applicability of Group Technology models and clustering techniques of the industrial engineering and operation research community to the partitioning problem of electronic circuits is examined. The problem is shown to be NP-complete, hence intractable within most modern computing environments. Characteristics of the solution are outlined and a grouping heuristic algorithm is discussed. We derive lower bounds on the objective function for any set of constraints on pairs of gates that must be in the same chip. The lower bounds and the grouping heuristic procedure are used to develop a branch and bound algorithm. Finally, computational results are given for four test problems.

11.1 INTRODUCTION

11.1.1 THE ROLE OF CLUSTERING (GROUPING) IN DESIGN

In Chapter 5, we introduced the notion of a metric space. If a space is metric, then one can calculate a *distance* between any two entities in the function or attribute spaces. Defining a distance metric will enable to group entities (e.g. artifacts, structures, or anything else) into classes that reflect commonality of some properties (e.g. structural attributes, functional attributes). Such a *grouping* or *clustering* method works as follows:

1. for each entity, define the properties one cares about and be able to give numerical values for each property;
2. create a vector of length n with the n numerical values for each entity to be classified;
3. viewing the n -dimensional vector as a point in an n -dimensional space, cluster points that are near one another.

This procedure leaves the following issues open to variation:

1. the properties used in the vector;
2. the distance metric used to decide if two points are “close”;
3. the algorithm used to cluster.

To take a simple case, suppose we have two properties associated with car horns: *power consumption* and *thermal conductivity*, each ranging between zero and one. Suppose the graph of our data points looks like Figure 11.1. Assuming we use the “obvious” *Euclidean distance* metric between points, then this figure clearly suggests two groups, one with four members, one with three.

The synthesis process was defined in Chapter 5 as a mapping from the specifications properties to design description properties. Synthesis generates several candidate designs that are expected to satisfy the design specification. In contrast, *analysis* is concerned with the process of inferring potential functionality from artifact structure. The mapping from the specifications properties to the artifact description (i.e., synthesis) is often the “inverse” of the mapping from the artifact description to the specifications properties (i.e., analysis). In most domains, high quality synthesis knowledge does not exist (in contrast to analysis knowledge, which is well developed within engineering disciplines. However, synthesis is the most important process in design, as it deals with the creation of new artifacts.

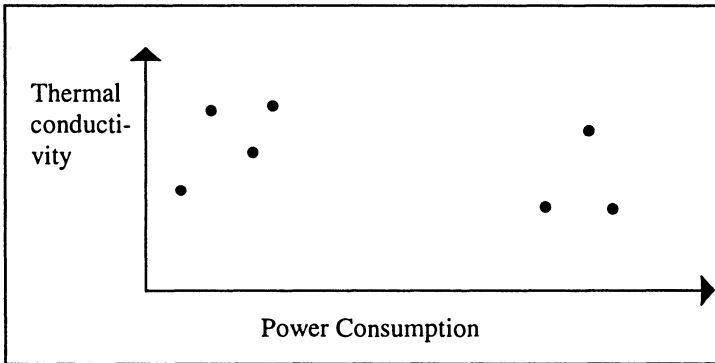


Figure 11.1 A simple Clustering Problem

Grouping or clustering algorithms can be used to support the synthesis process in several ways:

Learning Synthesis Knowledge

For relatively mature disciplines, examples of existing artifacts are available. Since these artifacts are the products of exercising synthesis knowledge, they implicitly embed this knowledge. Grouping algorithms provide a means for the acquisition of synthesis knowledge to support the synthesis process. The assimilation of design

examples into knowledge can be used to synthesize candidate designs for given specifications.

Grouping design examples may work as follows: the description of artifacts is limited to a fixed and pre-specified list of property-value pairs. This list is called the *artifact description*. The representation of the *specification* comprising the requirements and constraints for the artifact to be synthesized is also limited to a fixed, pre-specified list of property-value pairs. The grouping algorithm accepts a stream of examples described by a list of property-value pairs. Then, a distance metric used to decide if two designs are “close” is defined. For example, in the work reported in [30, 31], the proposed distance metric measures the expected number of property-pairs that can be guessed correctly by grouping two designs. What remains is the algorithm by which the groups (clusters) are created, given the metric used. For example, a typical strategy is to adopt a *greedy algorithm*. This means that the algorithm starts with ω groups, one for each design example. It then combines the two groups that result in a minimal loss of the distance metric, and repeats until the desired final number of clusters is reached.

To illustrate how a mapping from the specification, represented by several properties, to the full description of the artifact in terms of a much larger number of design description properties is created (i.e. synthesis), assume that a new design problem (specification properties) is introduced. The synthesis system may try to accommodate it into the existing classification. At this stage synthesis should terminate and candidate designs should be returned.

To take a particular example, consider the ECOBWEB algorithm reported in [31]. The following review is taken from [31]. ECOBWEB, an enhanced version of COBWEB [30], acquires synthesis knowledge and uses it to synthesize new bridges. ECOBWEB creates a classification from design examples represented by lists of property-value pairs. It has several operators that build the classification from examples. Learning and synthesis progress by one-step look-ahead search in the space of classifications directed by an evaluation function to select the best operator. The evaluation function, called *category utility*, evaluates a classification of a set of designs into mutually-exclusive classes C_1, C_2, \dots, C_n . When a new design is introduced, ECOBWEB tries to accommodate it into the existing classification by performing one of five operators that maximizes the value of the category utility function. In this approach, ECOBWEB retrieves a pre-determined number of candidate designs from the classification. The candidates are complete descriptions of previously designed bridges.

Grouping and Uncoupled Design

Learning synthesis knowledge via a grouping algorithm as described above restricts the scope of designs to those with fixed structure. Therefore, the design of artifacts that are described via graphs, such as layouts, cannot be supported under the assumption that artifacts and their specifications are described by (finite) lists of property-value pairs. However, grouping methods can also be beneficial in

synthesizing artifacts that are described via layouts. As discussed in Chapters 6, 10, the aim of a good design is to *uncouple* the functional requirements so that each design parameter (e.g. component or subsystem) affects only one set of functional requirements. Uncoupling means that each subsystem exhibits a minimum number of overlapping functional requirements. For example, electrical circuit partitioning, also called packaging or assignment, involves the transformation of a drawing of the logical circuit into sub-circuits, each of which is assigned to, or packaged in, a module (also called *integrated circuits*). For a given partitioning, the number of interconnections between modules is used as a measure of uncoupling (an efficiency measure). The smaller this measure is, the better the design is.

In this chapter, the applicability of grouping and clustering techniques to the partitioning problem of electronic circuits in order to achieve uncoupled solutions is examined.

Performance Metrics for Function Structures

If a space is metric, then the distance measure can be used to assign *values* (also called *performance metrics*) to each of the attributes or functions describing an artifact (see Chapter 5). Evaluation of structural attributes (e.g. a detailed circuit design at the physical level) is straightforward; performance metrics include, but are not limited to: size, weight, power requirements, efficiency, capacity for force generation and economic features. Unfortunately, precise performance metrics are difficult to articulate for designs that are completely described as function structures, e.g. a circuit design at the logical level. Grouping methods can be used to address the fundamental problem of articulating clear performance metrics for function structures by comparing the values of their associated structural attributes. For example, the functional requirements specified for a circuit can be fulfilled by more than one logical design using different sets of logical elements interconnected in different ways. The designer can select one of several alternative logical designs by comparing the values of their associated grouping into sub-circuits.

11.1.2 THE CIRCUIT PARTITIONING PROBLEM

The remainder of this chapter focuses on the physical design of *microelectronics circuits*. We apply the grouping approach to the *microelectronics circuit partitioning* problem, and use it to develop a branch and bound algorithm.

Circuit physical design involves the transformation of a logical design into a specific set of modules positioned within some specified frame, and the routing of the interconnections among the modules. This is illustrated in Figures 11.2 and 11.3. In Figure 11.2, a drawing of a logical circuit, called schematic, is represented. A physical design for the same circuit is represented in Figure 11.3. The blocks in the physical design of Figure 11.3 are the modules of the circuit which contain the circuit's logical elements. The portion of each module is very clear, and the routes of

all desired interconnections are drawn. It is now clear that circuit physical design consists of three major phases: *partitioning*, *placement*, and *routing* [28]. Appendix A provides background on electronic circuits and their design for the reader who is not familiar with the subject.

The partitioning phase, also called packaging or assignment, is the first in the physical design process. In the partitioning phase, the circuit is partitioned into sub-circuits, each of which is assigned to, or packaged in, a module (also called *integrated circuits*). For a given partitioning, the number modules, and the number of interconnections between modules are used as measures of efficiency. The smaller these parameters are, the better the design is. Less space is required for a smaller number of modules, i.e., higher space utilization; and a design with fewer interconnections between the modules is more manufacturable, reliable, and testable. For example, Figure 11.4 presents one feasible partitioning of the schematic of the logical circuit presented.

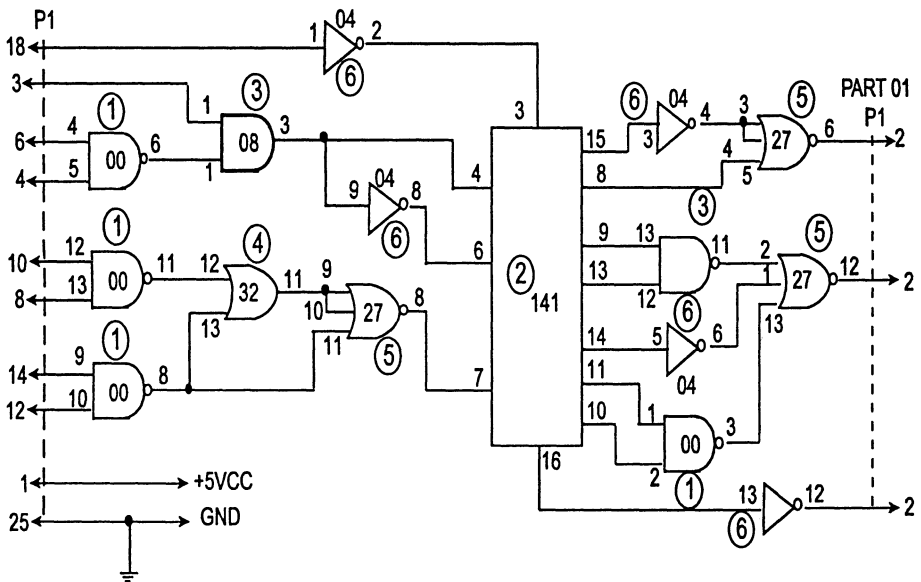


Figure 11.2 Drawing of A Logical Circuit, called Schematic

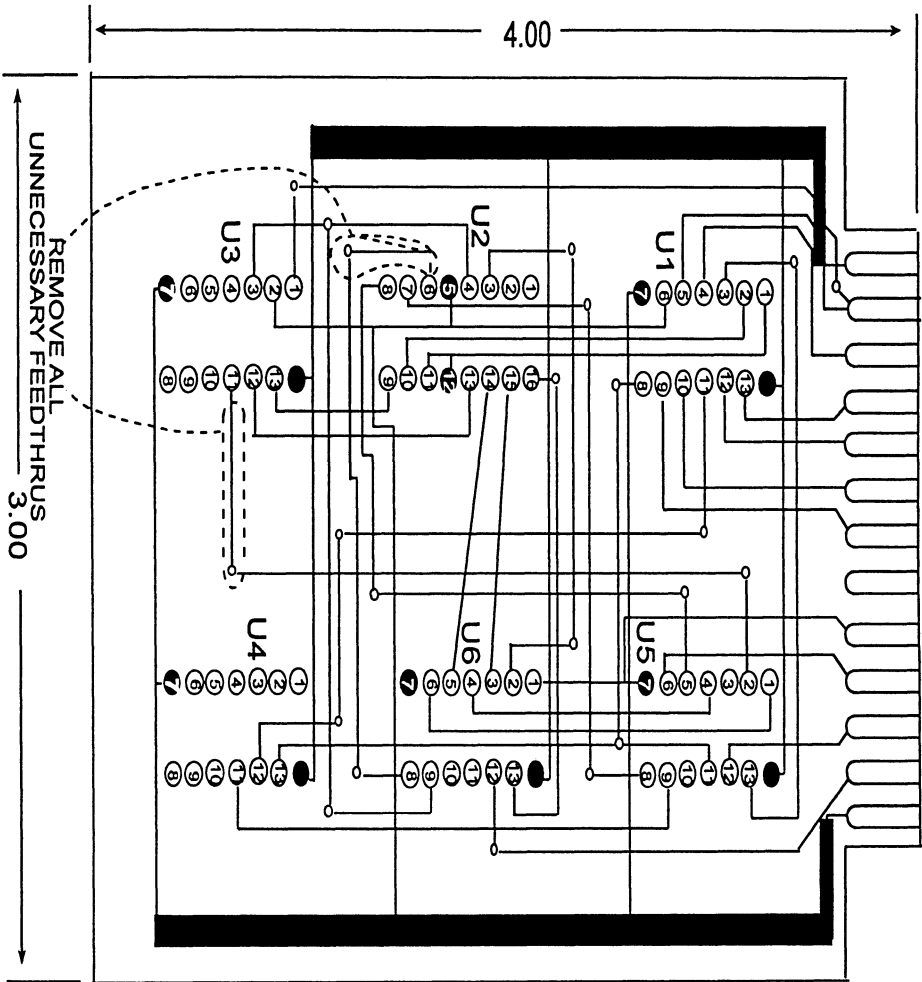


Figure 11.3 The Physical Design of the Logical Circuit Presented in Figure 11.2

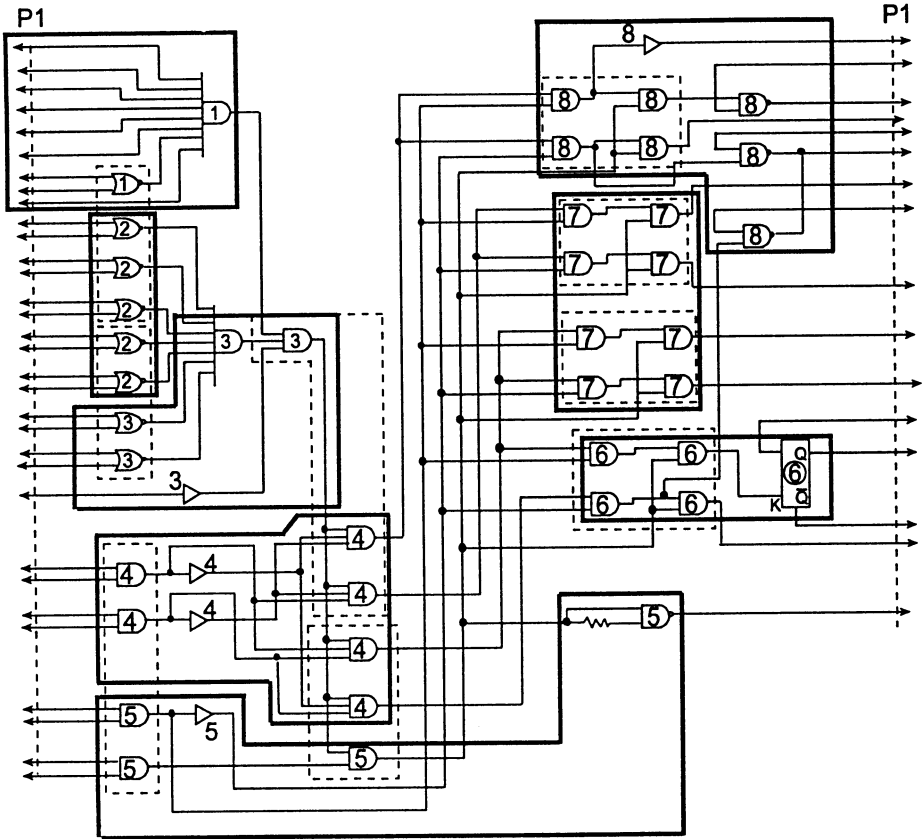


Figure 11.4 Feasible Partitioning for A Schematic

In this chapter, an optimization model is formulated for the *circuit partitioning problem* and an efficient grouping heuristic scheme is developed to solve large scale instances of this model. The proposed method can also be used to assist in the evaluation of the logical design. The functional requirements specified for a circuit can be fulfilled by more than one logical design using different sets of logical elements interconnected in different ways. The designer can select one of several alternative logical designs by comparing the values of their associated partitioning. Since the method presented here is geared to partition the logical design toward minimizing the number of on-board (i.e., between chips) connections, doing so is a step toward a design for routability, manufacturability, reliability, and testability.

The circuit presented in Figure 11.4 is specified at the gate level. However, the partitioning method presented here is capable of handling circuits at different levels of aggregations. A large and complex circuit can be partitioned top-down, and the modules at one design level can be further partitioned and internally designed at

parallel. Thus, in this chapter a 'circuit' can be a system level circuit, a board level circuit, or an integrated circuit (IC). For example, aggregation is used at the logical design phase, since million gate circuits are rarely designed at the gate level. Rather, previously designed functional elements are retrieved from cell libraries for the design of a new circuit. Also, most circuit consists of some well defined modules (e.g., ALU, memory) for which the designer needs no help in the partitioning. The parts of the circuit that provide the interface between the major modules are those that are hard to partition. The gate count of this 'interface logic' can be up to several tens of thousands in today's circuits.

In this chapter we assume that we are given a finite set of gates (components), a scheme of the electrical connections between them, and chips (termed also as packages or ICs) of different types. Associated with each gate is a list of the input/output pins (*I/O* pins). *I/O* pins that have a *common* electrical signal (thus connected by the same electrical wiring) are assigned to a particular *signal's net* type. The *I/O* pins connected by the net's signal are the *net's terminals*. A gate may be connected to more than one signal, i.e., a gate may belong to several nets. Underlying the problem formulation are the following assumptions: (1) the gates are assigned to different chips such that each chip has a finite capacity for different types of nets that reside in it. This constraint is placed to ensure that space limits are not violated, (2) costs are incurred when a net is included in a chip, since a pin is required for each net that partly resides in it, and (3) routing on the chip is cheaper and more reliable than routing on the next level of packaging. The conclusion of the previous discussion (see also Appendix A) is that minimizing the on-board (i.e., between chips) connections will result in a more modular, routable, reliable, testable, and manufacturable design. The number of on-board interconnections is thus used in the partitioning problem of this chapter as an effective measure of the design quality. The simple way to model the board level circuit is to include a connection between each pair of chips holding terminals of a net. The results of the packing efforts may be considered as an aggregate description of the schematic in which all the gates packed together are represented by a single chip, while on-board connections are used to connect net terminals packed in different chips. The ICs and on-board connections then form a board level circuit with board level nets.

Physical board design is similar to many design problems addressed in the Industrial Engineering (IE)/Operations Research (OR) literature. Yet, the applicability of IE/OR techniques are rarely used in the area of electronic circuits. In this chapter, the applicability of Group Technology models and techniques of the IE/OR community to the partitioning problem of electronic circuits is examined. Group Technology research of the IE/OR community has mainly been devoted to the problem of finding appropriate groups for cellular manufacturing [25, 23]. The problem is often cast in terms of grouping machines into cells such that the production of all components can be accomplished by machines in a single cell. Often, this is not possible unless machines can be included in more than one cell or unless components are moved between cells during production. In terms of our problem, components in the broader Group Technology literature are analogous to nets and machines are analogous to logic gates. Moving a component between cells

during production is analogous to connecting net's terminals packed in different chips. A key facet of our problem that is often ignored in the Group Technology literature is that there is a limit on the number of different net types that can be included in a chip. This is analogous to having a limit on the number of different machines that can be included in a cell. Harhalakis, Nagi, and Proth [9] have recently proposed a heuristic approach to cell formation that reflects this constraint.

Several approaches exist to group the machines into cells, such as (1) the application of clustering techniques to a part-machine matrix [12, 13, 3, 4, 17, amongst other], (2) the use of optimization-based approaches [e.g., 15, 14, 24], (3) graph theoretic approaches [e.g., 1, 21, 26], (4) simulated annealing [9, 10, 16], genetic algorithms [20], neural networks [6, 11] as well as the use of learning methods to minimize the traffic between cells [5]. A literature review summarizing these approaches is given in [25].

Recently, Daskin et al. [27] addressed the problem of assembling printed circuit boards (PCB). For a thorough review on automated process planning for PCB assembly and list of references, related to the many possible strategies for managing PCB assembly resources, see [18]. PCBs assembly involves mainly the insertion and soldering of electrical components (chips) into printed circuit boards. We assumed in [27] that we are given a number of different PCB types that are to be produced on a single machine with limited component staging capacity. Associated with each PCB type is a list of the component types that must be inserted into the PCB. The problem we addressed was to determine the subset of components in each group. The objective is to minimize a weighed sum of the total number of passes required by all PCB types; and the total of the subset cardinalities. We provided conditions under which it is optimal to load each PCB exactly once. When it is both feasible and optimal to load each PCB on the machine only once, we can also speak of grouping PCBs since a group of PCBs then implies a group of components.

The top design level, in which the whole board is designed, is the subject of this chapter. In this chapter, we show that PCB assembly and circuit physical design are similar, and only slight modifications may be needed to convert tools from one to the other. In particular by identifying 'PCBs' as 'gates' and 'components' as 'signal's nets', the partitioning phase of the physical design process is addressed using the methods presented in [27].

The remainder of the chapter is organized as follows. In Section 11.2 we formulate the problem as an integer linear programming problem. Unlike most of the diagonalization and similarity based approaches, the number of different types of nets that can be included in a chip (or equivalently the number of machines that can be in a cell) are limited. In Section 11.3 we show that the problem is NP-complete. In Section 11.4 we outline a clustering heuristic solution procedure. In Section 11.5 we derive bounds on the objective function. Using these bounds we develop a branch and bound algorithm that is shown to be effective at optimally solving small to moderate sized instances of the problem. In Section 11.6 we summarize our computational experience with the branch and bound algorithm. Section 11.7 concludes the chapter.

11.2 MATHEMATICAL FORMULATION

To formulate the problem of grouping logic gates in different chips, we define the following indices:

- i = index of net types ($i = 1, \dots, I$)
- j = index of gates ($j = 1, \dots, J$)
- g = index of chips ($g = 1, \dots, G$)

We define the following inputs:

- s_i = the cost incurred when the net type i is included in a chip
- L = the maximum number of net types allowed in a chip
- $a_{ij} = 1$, if net type i is associated with gate j ; 0, otherwise
- N_j = the set of indices of net types i required by gate j
 $= \{ i \mid a_{ij} = 1 \}$
- O_j = the set of indices of net types i associated *only* with gate j
 $= \{ i \mid a_{ij} = 1 \text{ and } a_{ij'} = 0 \ \forall j' \neq j \}$
- C_j = the set of indices of net types i associated with gate j and
 at least one other gate
 $= \{ i \mid a_{ij} = 1 \text{ and } \exists j' \neq j \text{ such that } a_{ij'} = 1 \}$

The net cost, s_i , will be incurred for each chip in which net type i is included. We assume that the costs s_i are strictly positive.

We define the following decision variables:

$$X_{ijg} = \begin{cases} 1, & \text{if net type } i \text{ of logic function } j \text{ is in package } g; \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{jg} = \begin{cases} 1, & \text{if logic function } j \text{ is packed in package } g; \\ 0, & \text{otherwise} \end{cases}$$

$$Z_{ig} = \begin{cases} 1, & \text{if net type } i \text{ is in package } g; \\ 0, & \text{otherwise} \end{cases}$$

Note that if net type i is partly resides in chip g , net type i can be connected to any gate j that is associated with it. With this notation, the *circuit-partitioning* problem may be formulated as follows:

$$\text{Minimize } \sum_i s_i \sum_g Z_{ig} \quad (1)$$

$$\text{Subject To: } \sum_g X_{ijg} = 1 \quad \forall i \in N_j, \forall j \quad (2)$$

$$Z_{ig} \geq X_{ijg} \quad \forall i \in N_j, \forall j \forall g \quad (3)$$

$$Y_{jg} \geq X_{ijg} \quad \forall i \in N_j, \forall j \forall g \quad (4)$$

$$\sum_i Z_{ig} \leq L \quad \forall g \quad (5)$$

$$\sum_g Y_{jg} = 1 \quad \forall j \quad (6)$$

$$X_{ijg} \in \{0, 1\} \quad \forall i \in N_j, \forall j \forall g \quad (7)$$

$$Y_{jg} \geq 0 \quad \forall j \forall g \quad (8)$$

$$Z_{ig} \geq 0 \quad \forall i \forall g \quad (9)$$

The inner sum in the objective function (1) equals the number of chips holding terminals of a particular net, i.e., the inner sum counts the number of on-board interconnections associated with a particular net. It then follows that the objective function (1) is for minimizing the positive weighted count of on-board interconnections. We assume that a cost is incurred for each net type in each chip (whether or not the net is in any other chip). Thus, even if a net is included in two chips, the objective function counts the net's cost a second time. Constraint (2) stipulates that each net - gate combination be assigned to exactly one chip. Note that it does not require that each net be in only one chip. Constraint (3) states that if net i of gate j is in chip g ($X_{ijg} = 1$), then net i must be assigned to chip g ($Z_{ig} = 1$). Similarly, constraint (4) states that if net i of gate j is in chip g , then gate j must be packed in chip g ($Y_{jg} = 1$). Constraint (5) is the capacity constraint for the number of nets types permitted in a chip. Constraint (6) states that each gate be assigned to exactly one chip. Constraint (7) is the integrality constraint on the assignment variables X_{ijg} . Because of constraints (3) and (4), the fact that the objective function is a minimization with positive coefficients, and the integrality constraint on the assignment variables X_{ijg} , the decision variables Y_{jg} and Z_{ig} need only be constrained to be non-negative as shown in (8) and (9). To prove this result, let us consider the following cases: (a) If $X_{ijg} = 1$ for some j , then $Z_{ig} = 1$ is a feasible solution that minimizes the objective function (1); (b) If $X_{ijg} = 0$ for every j , then $Z_{ig} = 0$ is a feasible solution that minimizes the objective function (1). Similar arguments hold for Y_{jg} .

Maimon and Shtub [17] provided a non-linear integer programming formulation of the *circuit-partitioning* problem. By introducing the variable X_{ijg} , we obtain a linear integer programming formulation.

Other variations on the basic model may also be formulated. For example, formulation (1)-(9) does not force or induce small chips to be combined. However, two small chips may be identified which have no nets in common and which could be merged into a single chip without violating the capacity limit on the number of different nets in a chip. To model such problems, we can introduce a cost of using an

additional chip. In what follows, when we refer to the *circuit-partitioning* problem, we are referring to formulation (1)-(9).

11.3 PROPERTIES OF THE CIRCUIT-PARTITIONING PROBLEM

The main result in this section concerns the computational complexity of the *circuit-partitioning* problem. Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources - such as computation time, space and hardware size - required to solve problems via digital algorithms. A *problem* is a collection of *instances* that share a mathematical form but differ in size and in the values of numerical constants in the problem form. In general we convert optimization problems to *decision problems* by posing the question of whether there is a feasible solution to a given problem with objective function value equal or superior to a specified threshold. The notion of "easy to verify" but "not necessarily easy to solve" decision problems is at the heart of the Class *NP*. Specifically, *NP* includes all those decision problems that could be polynomial-time solved if the right (polynomial-length) "clue" or "guess" were appended to the problem input string. An important subclass of *NP* problems are referred to as *NP-complete* or Non-deterministic Polynomial time Complete problems [8]. The CPU time required to solve an *NP-complete* problems, based on known algorithms, grows exponentially with the "size" of the problem. There exists no polynomial time transformations for *NP-complete* problems, nor are there any polynomial time algorithms capable of solving any *NP* problems. The potential to solve *NP* and *NP-complete* problems depends on the availability of certain heuristics. One problem *polynomially reduced* to another if a polynomially bounded number of calls to an algorithm for the second will always solve the first. A problem Ω is shown to be *NP-complete* by polynomially reducing another already known *NP-complete* problem to Ω .

We now show that the *circuit-partitioning* problem is *NP-complete* in the strong sense. We do this by showing that the *3-PARTITION* problem, which is *NP-complete* in the strong sense [8, pp. 96-100] polynomially reduces to a special case of the *circuit-partitioning* problem. The *3-PARTITION* decision problem is defined as follows:

INSTANCE: An integer B , $3m$ integer numbers d_l satisfying (i) $B/4 < d_l < B/2$ and (ii) $\sum_l d_l = mB$.

QUESTION: Does there exist a partition of the d_l into m disjoint sets P_1, P_2, \dots, P_m such that $\sum_{l \in P_g} d_l = B$ for each set P_g ?

If the answer to the *3-PARTITION* decision problem is yes, each set P_g will contain exactly 3 elements (because of the bounds (i) on the values d_l). We can define the *circuit-partitioning* decision problem by simply asking whether or not a solution to the *circuit-partitioning* optimization problem exists with a value less than or equal to

some input constant Z (recall that this problem is computationally equivalent to the original problem).

Theorem 1: The *circuit-partitioning* problem is NP-complete in the strong sense.

Proof: The proof is given in Appendix B.

From the proof it can be shown that the *circuit-partitioning* problem remains NP-complete even when (1) each chip is constrained to have 3 or fewer PCBs, and/or (2) when all net packaging costs are identical.

We note that if each group is constrained to have 2 or fewer gates, the problem can be solved in polynomial time using a minimum weighted matching algorithm. The graph for this problem is constructed as follows (an example is presented in Figure 11.5):

Example Net/Gate Requirement Matrix (with net capacity, $L = 4$):

Net	Gate			
	1	2	3	4
1	1	1	0	0
2	1	0	1	0
3	0	1	0	0
4	1	0	1	0
5	0	0	1	1

The Associated Graph Representation:

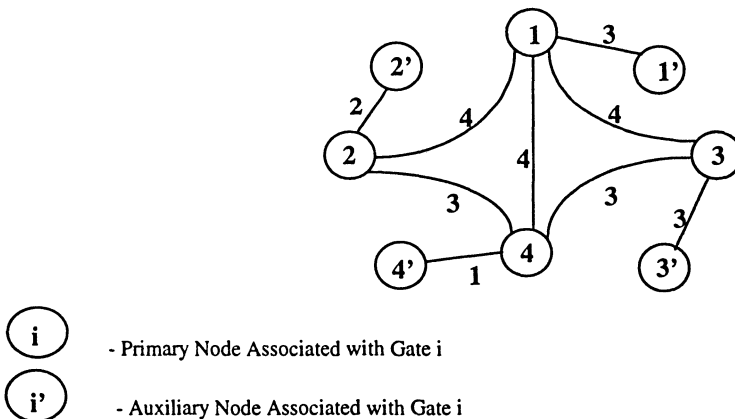


Figure 11.5 Transforming A Circuit-partitioning Problem into A Minimum Weighted Matching

We associate a primary node with each gate. A link connects each pair of primary nodes associated with gates that can be included in the same chip (i.e., gates whose combined net requirements do not exceed the capacity of the chip). The cost of each such link is the total net packaging cost of the nets that are required by the two gates. In addition, we create an auxiliary node for each gate. A link is added to the graph between the auxiliary and primary nodes for each gate. The cost associated with such links is simply the total net packaging cost of the gate's nets. (Such links allow for the possibility of having chips that include only one gate.) The solution to the minimum weighted matching problem (which can be obtained in $O(J^3)$ time [19] identified the chips that should be formed to include pairs of gates as well as the chips that should be formed to include only a single gate.

11.4 A GROUPING HEURISTIC FOR THE CIRCUIT-PARTITIONING PROBLEM

Since the general *circuit-partitioning* problem is NP-complete, a heuristic solution algorithm is likely to be needed. This section outlines a simple heuristic that was used to find feasible solutions and was part of the branch and bound algorithm described in Section 11.5. Branching in the branch and bound algorithm is based on whether two gates must be in the same chip or must be in different chips. We note that a set of gates that, at some point of the branch and bound algorithm, must all be included in the same chip may be thought of as a mega-gate whose net requirements are equal to the union of the sets of nets (N_j) required by the gates that must be packed together (we refer to them simply as gates). If the cardinality of the union of these sets exceeds the capacity of the chip, there is no feasible solution satisfying the constraints that forced the gates to be packed together. We assume that the number of nets required by any mega-gate does not exceed the capacity of the chip. In fact, the branch and bound algorithm checks that this condition is satisfied before calling the heuristic algorithm.

The heuristic sequentially constructs groups of gates (and nets), and is composed of the following steps (see Figure 11.6):

Step 1 (Identify all mega-gates): The algorithm begins by identifying mega-gates based on the current set of constraints (if any) for pairs of gates that must be included in the same chip. After identifying the mega-gates, we will have a number of mega-gates and a number of gates which are not constrained to be packed with any other gates.

Step 2 (Begin a new chip): Once all mega-gates are identified, a chip is initiated with the unassigned gate (or mega-gate) that requires the smallest number of nets.

Step 3 (Identify and assign the unassigned gates): Gates (or mega-gates) are added to the emerging chip, beginning with the gate (or mega-gate) with the smallest number of additional required nets, until either (1) no unassigned gates (or mega-gates) remain, or (2) there are no more unassigned gates (or mega-gates) which can feasibly be assigned to the emerging chip. Feasibility is based on: (1) the chip capacity, L , and

(2) the set of constraints (if any) that preclude pairs of gates from being packed in the same chip. If inclusion of a candidate gate (or mega-gate) would violate any of these constraints for a gate already included in the emerging chip, the candidate gate (or mega-gate) is not included in the chip. If inclusion of the candidate gate (or mega-

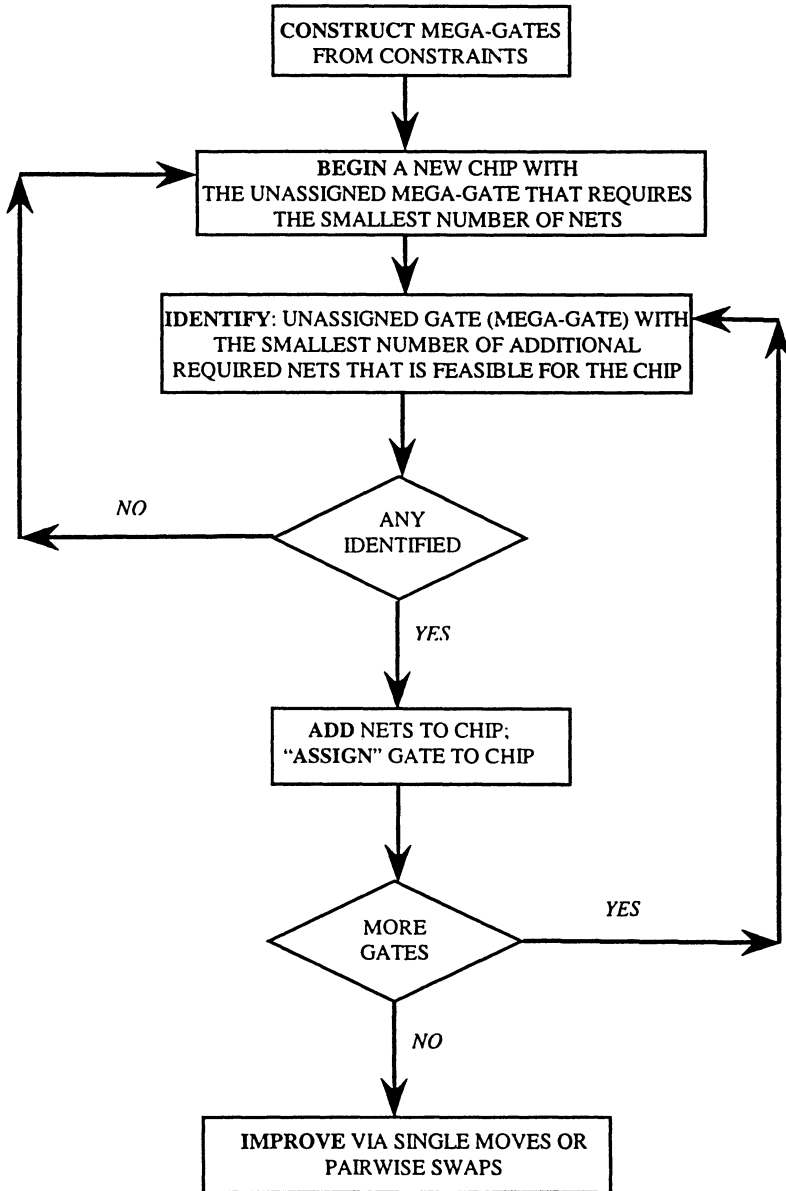


Figure 11.6 Flowchart of Heuristic Algorithm

gate) in the emerging chip would not have violated the chip capacity, the gate (or mega-gate) with the next smallest number of additional required nets is considered for inclusion in the emerging chip (i.e., repeat **Step 3**). If no gate (or mega-gate) may be added to the emerging chip and additional unassigned gates (or mega-gates) remain, a new chip is initiated (go to **Step 2**). If all gates (or mega-gates) have been assigned to a chip, **STOP**.

After all gates are included in some chip, we have a feasible solution to the problem. The heuristic algorithm includes the capability of performing single gate moves and pairwise swaps of gates. A single gate move consists of removing a gate from its current chip and evaluating the cost savings that would result from including the gate in another chip. If a feasible move of this sort can be identified and if the move results in a cost savings, the move is executed. Similarly, a pairwise swap consists of removing two gates from different chips and evaluating the cost savings that would result from including each gate in the chip in which the other was included. If a feasible swap of this sort can be identified and the swap reduces the total cost, the swap is executed. The heuristic algorithm alternates between attempting to perform a move and trying to perform a swap. Improvements continue until neither a feasible cost saving move nor a feasible cost saving swap can be identified.

11.5 A BRANCH AND BOUND ALGORITHM

In developing a branch and bound algorithm the following issues need to be resolved:

- (1) the variable(s) on which to branch and the means of identifying the next variable on which to branch at each node of the tree;
- (2) the selection of the node in the tree from which to branch;
- (3) means of bounding the solution at each node; and,
- (4) when to apply upper bounding heuristics.

Each of these issues is briefly addressed in the subsections below. First, however, we summarize a domination property that allows us to identify pairs of gates which must be in the same chip in an optimal solution.

A Domination Property

If $N_j \subseteq N_k$, then at least one optimal solution will have gate j and gate k in the same chip. In this case we say that gate k dominates gate j .

To prove this property, suppose we have an optimal solution in which gate j and gate k are in different chips, which we refer to as chips 1 and 2 respectively, without loss of generality. Since each gate requires exactly one group of nets and since $N_j \subset N_k$, moving gate j from chip 1 to chip 2 will not increase the number of nets required to be packed in chip 2. Thus, the objective function can not increase by such an exchange. If the objective function were to decrease as a result of such an exchange,

the original solution could not have been optimal. Note that this domination property also holds if we replace N_k by \mathcal{N}_k (the set of net types i used by gate k or any gate forced to be in the same chip as gate k).

This property allows us to force dominated/dominating pairs of gates together, thereby reducing the effective size of the problem by branching on these pairs of gates at the beginning of the branch and bound tree. It also allows us to fathom nodes of the tree for which this property is violated. Note that the optimal solution may involve producing gates k and h in different chips while $N_j \subseteq \mathcal{N}_k$ and $N_j \subseteq \mathcal{N}_h$.

The domination property implies only that at least one optimal solution involves producing gate j with either gate k or gate h . Thus, in using this property in the branch and bound algorithm, care must be taken not to fathom a node in which gate j is forced not to be with gate h , for example, if gate j is already forced to be with gate k .

Branching Rules

Branching was done on whether or not a pair of gates must (or must not) be in the same chip. In all cases, we branched from the rightmost unfathomed node in the tree.

To determine the next pair of gates on which to branch, we computed the following statistic for each pair of gates (provided gate j is not dominated by gate k):

$$\begin{aligned}\phi_{jk} &= \text{the number of net types used by both gate } j \text{ and gate } k \\ &= \sum_i a_{ij} a_{ik}\end{aligned}$$

If gate j is dominated by gate k , we set $\phi_{jk} = M$, where M exceeds the number of nets in the problem. Note that ϕ_{jk} need only be computed once, before the beginning of the branch and bound procedure. The ϕ_{jk} values were sorted into decreasing order. At any node in the branch and bound tree, the pair of gates on which the algorithm branched was the pair (j', k') with the largest ϕ_{jk} value such that j' and k' were not either forced to be in the same chip or forced to be in different chips by constraints that had been imposed further up the tree. Setting $\phi_{jk} = M$ if gate j is dominated by gate k in essence forces this pair of gates to be in the same chip since (i) the algorithm will branch on such pairs of gates before any other pairs are processed and (ii) the branch in which they are constrained to be apart will be fathomed by the domination property outlined above.

Bounding the Solution at Each Node

A lower bound on the total cost can be obtained by multiplying a lower bound on the number of chips into which each net type i must be included by the net i packaging cost, s_i . At each node of the tree, four approaches were used to bound the number of

chips in which net i must be included. The **first bound** is given by the total number of net types needed by all gates (or mega-gates) that require net type i divided by the maximum number of nets that can be in a chip, L , rounded up to the next larger integer (see Appendix C.1). The **second bound** is given by the total number of gates ($\in \mathcal{M}_i$) which, at some node in the branch and bound tree, are mutually forced to be in different chips (see Appendix C.2). The **third bound** involves constructing a graph, $G_i(V_i, E_i)$, in which we create a vertex in the set V_i for every mega-gate or unconstrained gate (in the sense of not being forced to be in the same chip as another gate) that requires net type i . We create an edge between two vertices j and k in the graph if any gate associated with vertex j is constrained to be in a different chip from any gate associated with vertex k . The chromatic number of this graph, $\gamma(G_i)$, (the minimum number of colors needed to color each vertex such that no two vertices that are connected by an edge have the same color) is a lower bound on the number of chips in which net type i must be included, based on the constraints at that node of the branch and bound tree on pairs of gates that can and can not be in the same chip. Unfortunately, the chromatic number problem is itself an NP-complete problem. However, a number of lower bounds on the chromatic number have been proposed. All are functions of the number of vertices and edges in the graph. The best of these is the lower bound proposed by Ersov and Kozuhin [9]. See Appendix C.3 for details. The **fourth bound** is based on the size of the maximum clique (a clique in a graph G is a subgraph of G that is complete, i.e., each pair of vertices is connected by an edge) in a graph G_i whose construction was outlined above. If $\rho(G_i)$ is the size of the maximum clique in graph G_i , then $\gamma(G_i) \geq \rho(G_i)$. Unfortunately again, the problem of finding the maximum clique in a graph is also NP-complete [8]. However, in $O(n^3)$ time we can check whether or not a clique of size 3 exists (in which case the fourth bound is set to 3; if not it is set to 0) simply by enumerating all possible combinations of 3 nodes and checking whether or not all three nodes are connected to each other. The actual lower bound on the number of times net type i must be packed is the maximum of the four bounds outlined above.

Fathoming Nodes

Nodes in the tree can be fathomed in one of three ways. First, if the constraints that force pairs of gates to be in the same chip create a mega-gate whose net requirements exceed the capacity, L , of the chip, no feasible solution exists and the node may be fathomed. Second, if the net packaging cost based on the lower bounds on the number of times each net must be packed is equal to or greater than the value of a known solution, the node can be fathomed. Third, a left node (in which gates k and h are constrained to be in different chips) may be fathomed if (i) either k dominates h and h is not already dominated by some mega-gate created further up the tree or (ii) vice versa. Similarly, a right node (in which gates k and h are constrained to be in the same chip) may be fathomed if the newly created mega-gate dominates some gate j

which is forced not to be in the same chip as either k or h and gate j is not already dominated by some mega-gate created further up the tree.

Application of the Heuristic Algorithm

The heuristic algorithm was applied (i) at the root node, (ii) at nodes created by branching on a dominated/dominating gate pair, and (iii) when the list of ϕ_{jk} values was exhausted. If the list of ϕ_{jk} values is exhausted at some node in the branch and bound tree then there is no pair of gates that are not already either constrained to be in the same chip or that are constrained to be in different chips. In that case, the heuristic algorithm simply evaluates the cost of the solution specified by the constraints that have already been imposed. That means that the heuristic solution at the node at which we are trying to branch may be treated as a lower bound on the cost at that node in the tree. Since the lower bound then equals or exceeds the value of the best known solution, the node may be fathomed.

11.6 COMPUTATIONAL RESULTS USING THE BRANCH AND BOUND ALGORITHM

In this section we outline results obtained from testing the branch and bound algorithm using four data sets (DATA1, DATA3, CLUSTRA and CLUSTAX). In Table 11.1 is a summary of key features of the four data sets. DATA3 was constructed from DATA1 in two steps. First, we created a duplicate copy of each gate, thereby generating a problem with 16 gates. In the resulting matrix each net would be a shared net. Therefore, we removed one of the occurrences of each of the unique nets in DATA1 from the matrix generated by creating a copy of each gate. In removing these components, we alternated between removing the net from the original gate (numbered 1-8) and the generated gate (numbered 9-16). In this way, we created a problem which again had 43 unique nets, and 10 shared nets. For each row corresponding to a shared gate, the density of ones was the same in DATA1 and DATA3.

The size of the problems analyzed ranged from 8 gates and 53 nets (with only 10 shared nets) for DATA1 to 34 gates and 162 nets (with 103 shared nets) for CLUSTAX. The density of ones in the matrices ranged from 12.1% (CLUSTAX) to 18.4% (DATA1).

Results for DATA1 and DATA3

Table 11.2 is a summary of the key results for DATA1 and DATA3. We focus our attention on the data set DATA3 because it includes problems with larger size of the gate - net input matrix, which were the most difficult problems to solve using our

approach. The results for DATA3 are broken into two groups based on the chip capacity. Results for capacities 11-13 are reported separately from those for capacities 14-53 for two reasons. First, the solution times for the small capacity problems were all very large. Second, by separating the results for capacities in the range 14-53, we can more readily compare the results with those found for DATA1. The results for capacities 11-13 as well as the entire data set are shown in Table 11.2. The entire data set (capacity range 11-53) illustrates the effect of each capacity range. Larger capacity problems shift the results (e.g. solution time) towards significantly improved performance.

Table 11.1 Characteristics of Input Data Sets

DATA SET NAME		DATA1	DATA3	CLUSTRA	CLUSTAX
NETS					
	Unique	43	43	15	59
	Common	10	10	37	103
	Total	53	53	52	162
GATES					
	Total Number	8	16	26	34
	Number Undominated	8	16	17	22
GATES/ COMMON NET					
	Minimum	2	4	2	2
	Average	3.50	7.00	4.73	5.92
	Maximum	8	16	26	32
% OF 1s IN MATRIX		18.4	13.3	14.1	12.1
PACKAGING COSTS					
	All Gates	10	10	21	63
	All Nets	1	1	1	1
TESTED RANGE OF NET CAPACITIES		14 to 53	14 to 53	22 to 51	81 to 150

For DATA1, all solution times were under 1.05 seconds and the average was under 0.3 seconds; for DATA3 (with capacity range $14 \leq L \leq 53$), the maximum solution time was just under 1.76 minutes and the average was under 8.34 seconds. Table 11.2 also compares the number of nodes examined in the branch and bound tree to the number of solutions that would have had to be examined had total enumeration been employed. Even in the worst cases, only a minuscule fraction of the total number of solutions needed to be examined.

Finally, in over 77.5% of the runs, the initial heuristic solution was optimal when the move and swap improvement procedures were used. When they were not employed, this percentage dropped significantly, particularly for DATA3.

Table 11.2 Summary of Results for DATA1 and DATA3

DATA SET NAME		DATA1	DATA3	DATA3	DATA3
CAPACITY RANGE		14 to 53	11 to 13	14 to 53	11 to 53
SOLUTION TIME*					
	Minimum	0.10	462.58	0.22	0.11
	Average	0.30	639.72	8.34	52.39
	Maximum	1.05	986.90	105.24	986.90
NODES EVALUATED					
	Minimum	1	25,223	1	1
	Average	36	36,810	514	3,047
	Maximum	153	58,909	6,069	58,909
	Total Enumeration	2.68E+08	1.33E+36	1.33E+36	1.33E+36
% RUNS IN WHICH INITIAL HEURISTIC WAS OPTIMAL					
	w/ Move and Swap	92.5%	0.0%	77.5%	72.1%
	w/o Move and Swap	55.0%	0.0%	5.0%	4.7%
* Seconds on Zenith 386/16 Computer with an 80387 math coprocessor. Code was written in Turbo PASCAL version 5.5.					

Figure 11.7 is a plot of the percent of the total number of examined nodes that had to be explored before the optimal solution was found versus the chip capacity, L . Most (about 90%) of the computational effort is devoted to proving that a solution

obtained early in the branch and bound process is, in fact, optimal. In the few cases in which the percent shown in Figure 11.7 is large, the number of nodes examined is very small.

Figure 11.8 is a plot of the percent improvement in the solution that results from (1) the use of the move and swap algorithms and (2) the use of the branch and bound algorithm as a function of the chip capacity, L , for DATA3 (with gate - net matrix as defined above). Improvement percentages are measured relative to the optimal value. Most of the improvement of the initial heuristic solution was due to the move and swap procedures and not to the branch and bound algorithm. In many cases, particularly for larger chip capacities, the branch and bound algorithm was only used to confirm the optimality of the solution obtained from the initial heuristic followed by the move and swap procedures. Similar results were obtained for DATA1.

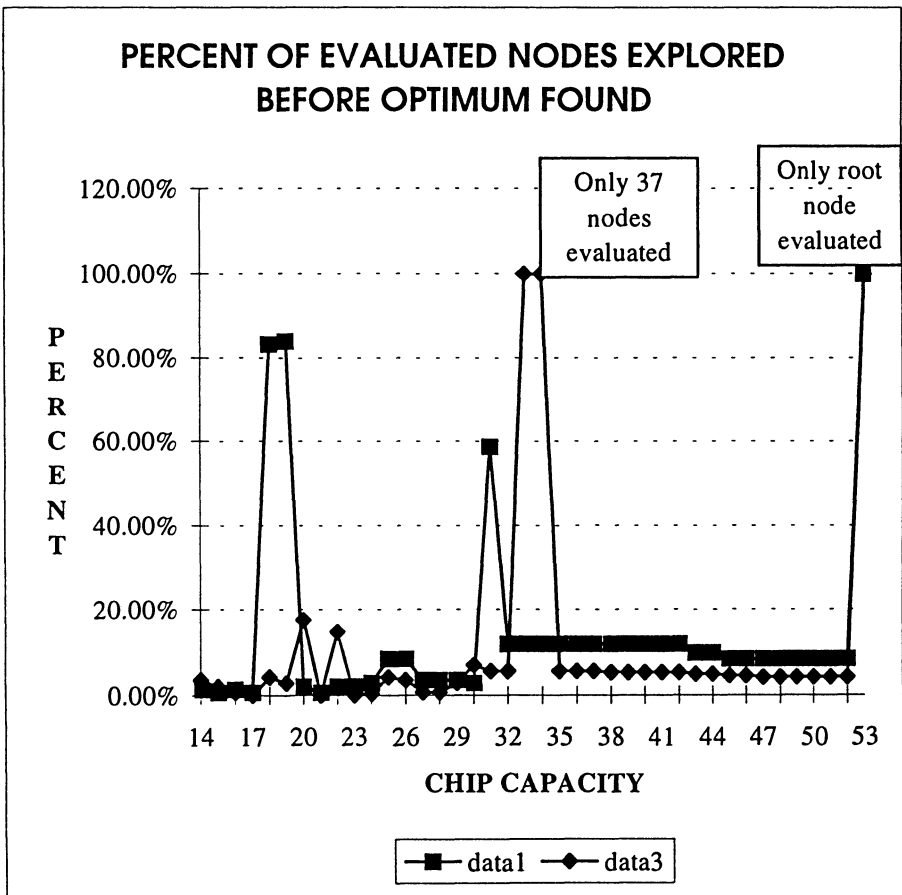


Figure 11.7 Percentage of Evaluated Nodes Explored Before Optimum Found

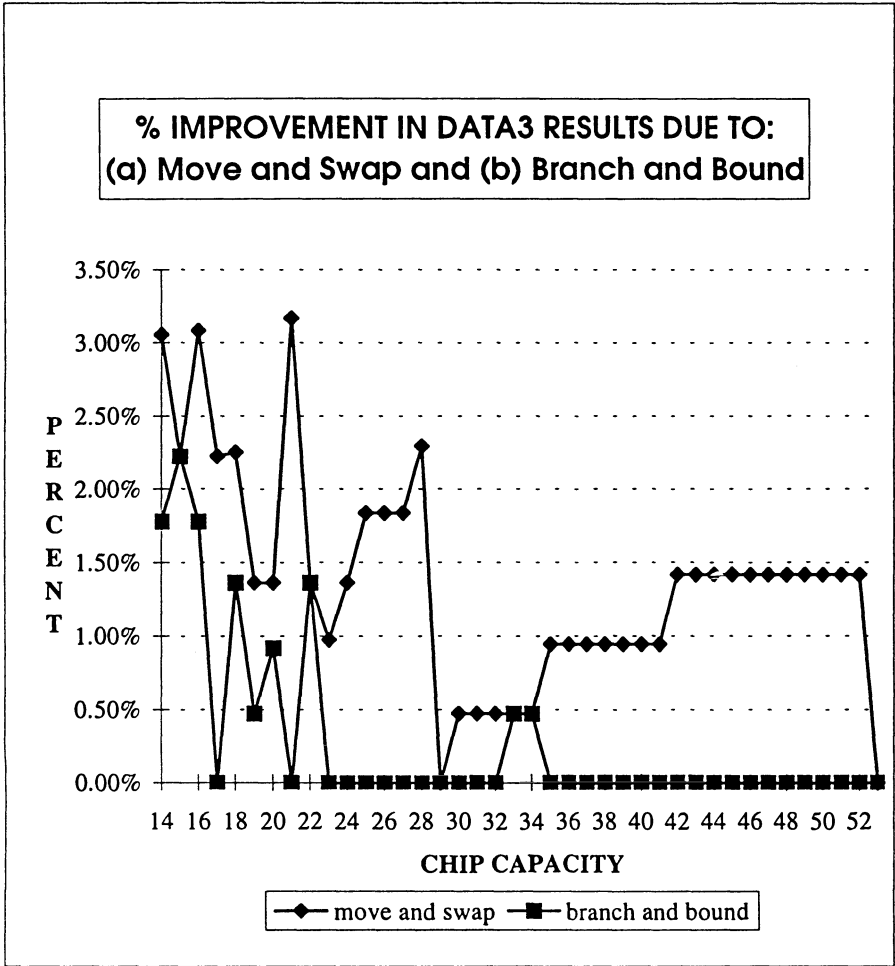


Figure 11.8 Percentage Improvement in DATA3 Results due to (a) Move and Swap and (b) Branch and Bound

Results for CLUSTRA and CLUSTAX

For CLUSTRA and CLUSTAX, in addition to solving the problems to optimality, we solved the problems by fathoming nodes in the tree whenever the difference between the best known upper bound and the lower bound was less than ϵ times the lower bound. Two values of ϵ were used (0.02 and 0.01) as shown in Table 11.3. Finally, instead of running each problem as an independent run, the best solution found for the problem with a chip capacity of L was taken as a feasible solution at the root node

of the branch and bound tree for the problem with a capacity of $L + 1$. This solution was compared to the heuristic solution (for a capacity of $L + 1$) and the smaller value was used as the upper bound at the root node of the branch and bound tree for the problem with a capacity of $L + 1$. Thus, successive runs for these data sets are not independent.

Table 11.3 Summary of Results for CLUSTRA and CLUSTAX

DATA SET NAME		CLUSTR A	CLUSTRA	CLUSTRA	CLUSTAX	CLUSTAX	CLUSTAX
ε - OPTIMALITY		0	0.01	0.02	0	0.01	0.02
SOLUTION TIME*							
	Minimum	0.22	0.33	0.27	50.10	8.84	1.15
	Average	96.05	17.19	3.12	2407.91	75.38	2.55
	Maximum	1239.28	336.37	33.45	26324.52	1,192.93	12.52
NODES EVALUATED							
	Minimum	1	1	1	259	27	1
	Average	2,410	417	47	21,279	616	7
	Maximum	34,857	8,905	783	243,353	10,885	85
	Total Enumeration	6.84E+97	6.84E+97	6.84E+97	7.55E+168	7.55E+168	7.55E+168
% RUNS IN WHICH BRANCH AND BOUND ALGORITHM NEEDED		96.8%	83.9%	67.7%	100.0%	100.0%	10.0%
PERCENT NET COST OVER MINIMUM							
	Minimum		0.00%	0.00%		0.00%	0.00%
	Average		2.62%	4.95%		3.04%	3.04%
	Maximum		8.20%	20.00%		11.23%	11.23%
* Seconds on Zenith 386/16 Computer with an 80387 math coprocessor. Code was written in Turbo PASCAL version 5.5.							

For both data sets, both the solution time and the number of nodes examined in the branch and bound tree increase dramatically as ϵ decreases. The time required to solve the problems optimally averaged under 2 minutes for CLUSTRA and just over 40 minutes for CLUSTAX. For $\epsilon = 0.01$, these times dropped to 18 and 75 seconds respectively. As expected, the percent of runs in which the branch and bound algorithm was needed (either to improve on the root node solution or to confirm its ϵ -optimality) decreased as ϵ increased. Finally, Table 11.3 reports the actual percent deviation from optimality of the net packaging cost in the objective function. The largest percentage deviations tended to occur at the largest chip capacities where the net packaging cost was the smallest. However, these problems also tended to be the easiest to solve to optimality. Thus, in practice it may be feasible simply to solve the large capacity problems optimally or with very small values of ϵ to reduce the deviation of the net packaging cost from its optimal value.

11.7 SUMMARY

The main purpose of this chapter is a demonstration of the applicability of the mathematical programming and group technology to circuit design, and in particular to the physical design of large scale electronic circuits. An optimization model is formulated and solution methods are developed and implemented for the partitioning phase of the physical design. We also address the fundamental problem of articulating performance metrics for function structures (i.e., *microelectronics circuits*) by comparing the values of their associated attributes (i.e., the number of on-board interconnections).

The efficiency in circuit design will become more crucial with the continuing rapid progress in microelectronics technology, while the scale of circuit integration will grow larger. Thus, the increasing of circuit complexity will render the construction of efficient designs more difficult. The more complex the circuit is, more concerns must be accounted for when the circuit is designed. This includes design for reliability, testability, and manufacturability. Finally, further development of the tools as described here will facilitate the integration of the currently separated algorithms into a computer integrated design tool.

APPENDIX A A BRIEF OVERVIEW OF MICROELECTRONICS CIRCUITS AND THEIR DESIGN [28]

For the reader who is not familiar with microelectronics circuits, a brief description of microelectronics circuit design follows.

A.1 MICROELECTRONICS CIRCUITS

Electronic circuits can be implemented in different ways: wired circuits, printed

circuit boards (PCBs), and integrated circuits (ICs). Wired circuits are common at the system level. The electrical connections of these circuits are established by wires. A printed circuit board is basically a plastic resin material coated with copper foil. In printed circuit boards the interconnections are established by copper lines attached (printed) to a solid or flexible (insulating) mattress. The conductive patterns on printed circuit boards are defined by lithography and selectively etching the copper. The integrated circuits and other discrete components are then fastened to the board by soldering. This is the final step in making the integrated circuits and the microelectronics devices they contain accessible. Such printed circuit boards are the hearts of computers and other large electronic systems. The components of the PCBs (especially discrete components) are usually packed in integrated circuits. The integration ranges from small and medium scale of integration (SSI/MSI) through large scale integration (LSI), to very large scale integration (VLSI). The integrated circuits are the chips mentioned in the chapter. Recently, custom and semi-custom designed (or custom specific) ICs have grown in popularity in order to better utilize chip space. These are usually components of large or very large integration that are specifically tailored to meet customer requirements.

A.2 THE GENERAL DESIGN OF PRINTED CIRCUIT BOARDS

The design of an electronic circuit consists of four major stages: (1) functional (logic) design; (2) Physical design; (3) test design and programming; and (4) manufacturing process planning.

The first stage is the *logical design*. In this stage the requirements and specifications that are provided to the designer are transformed into a detailed design at a logical level. The elementary functional elements of the circuit in the logical design and the interconnections between them are described using standard symbols in a chart called a *schematic* (see Figure 11.2). There are no scales and no consideration is given to component placement at this stage.

A given logical design can be implemented in more than one way. In the *physical design* stage, the board is configured to contain all the elements of the schematic and satisfy the functional specifications. In many cases, the logical designer is also responsible for part or all the physical design. The physical design consists of three main stages: (1) the elements of the schematic have to be packed in ICs (*'packing'* or *'partitioning'*); (2) each IC has to be positioned in a well-defined place on the board (*'placement'*). These positions will later be used to direct the assembly process; and (3) a detailed routing of the interconnections on the board, avoiding short circuits, must be specified (*'routing'*). This demarcation of the physical design stages is somewhat misleading, since all three stages are inextricably intertwined and are not distinct phases in the process of physical design. However, considering the very large scale of today's circuits, the above decomposition of the physical design is often used to reduce the computational intractability that characterizes the physical design task.

One product of the physical design stage is the artwork. The artwork describes the input for the fabrication in which the 'wires' are printed on the surface of the board.

In a multi-layer board, the 'wires' are printed on the layer of the board and holes are drilled if leaded components are used. Other products are the assembly drawing and the list of parts according to which the components are mounted on the board.

Turning the physical design into a physical board requires a detailed plan of the manufacturing process in order to achieve high productivity. This is because a clear trade-off exists between the sequence that completes each PCB type once it is loaded onto the machine (minimizing the number of times each PCB is loaded onto the machine), and a sequence that simultaneously loads all PCBs using the same group of components loaded onto the machines (minimizing the number of times each component type is loaded on the machine.) For a thorough review on automated process planning for PCB assembly and list of references, related to the many possible strategies for managing PCB assembly resources, see [18].

The complete design is used to generate the test program for the board. The logic design is used to derive the test patterns, while the physical design is needed to determine how to implement each pattern.

To summarize, the output of the board design efforts provides all the information required by subsequent stages, i.e., IC design and a detailed plan of the manufacturing process. The logical elements in the schematic are grouped into sets which are packed in different chips. This top design level, often called 'partitioning' or 'packaging,' is the subject of this chapter.

A.3 BOARD PHYSICAL DESIGN OBJECTIVES

Two measurable quantities that are commonly used to evaluate the quality of microelectronics circuits are the *wiring length* (interconnections), and *wiring congestion*, as will now be explained.

An important concern in circuit design is *routability* or wirability of the chip. The routability of a design is the extent to which the design circuit can be automatically routed. The total wiring length is a popular approximation of routability. The wiring length is to be minimized not only for higher wirability but also because it greatly affects time delays and power requirements. Local wiring congestion is also used to approximate routability, prohibiting congestion in excess of the maximum available by the wiring space in any point of the chip. If at any point the design requires more wires than allowed by the space available, the chip is unroutable. An upper bound on wire density can be calculated as in [29]. The actual objective is then to design the cell in order to either balance the wiring congestion, or to minimize the number of grid line crossings [29].

Routability is also of major importance at the board level since the wiring tracks are wider. In addition, routability is related to manufacturability. A board with high routability score usually requires fewer number of layers, and be simple to assemble. Thus, a higher production rate and lower costs are thus expected when a highly routable printed circuit board is implemented.

Reliability is of major concern too. At the board level the majority of the faults are *shorts* and *opens*. Opens are disconnection along tracks. Shorts occur when two

tracks -- that should not be connected -- are routed to cross each other. *Crossovers* are not allowed in a single layer single sided board. In other boards, *feed-thrus* (or *vias*) are used in order to prevent shorts circuiting the board. The number of vias is determined only after the board is completely routed. In *vias*, one track is fed through (via) a hole in the board to a different layer, thus preventing the contact between the crossing signals. However, feed-thrus are often the cause of opens, and thus should be minimized. Reliability is closely related to routability: a board with high routability score usually requires fewer feed-thrus.

Another important concern in circuit design is the design-for-testability concept. Functional grouping of the components into chips at the board level will reduce the complexity that is related to the laborious tasks of fault detection and fault diagnosis of the testing stage.

To summarize, a board with fewer and shorter interconnections is more routable and reliable, and less costly to manufacture, test, and maintain. Wiring congestion and length, are used to evaluate the quality of microelectronics circuits in the physical design stage.

APPENDIX B (PROOF OF THEOREM 11.1)

First, the *circuit-partitioning* problem is in NP since, given any solution (defined by values of the $O(I \cdot J^2)$ decision variables), we can check in polynomial time whether or not the constraints are satisfied and whether or not the objective function is less than or equal to some value Z . Note that there will be at most J chips, where J is the number of gates. This bound will be realized if each gate uses only unique nets (i.e., $O_j = N_j, \forall j$) and $L = 1$. Finally, we note that in practical problems the number of chips will usually be smaller than J .)

Next, we show that there is a polynomial transformation from the *3-PARTITION* problem to an instance of the *circuit-partitioning* problem, such that there is a solution to the *3-PARTITION* problem if and only if there is a solution to the *circuit-partitioning* problem.

For $j = 1, \dots, 3m$, (note that $J = 3m$ in this instance of the *circuit-partitioning* problem) let:

$$D_0 = 0 \text{ and } D_j = \sum_{m=1}^j d_m$$

Define the following sets of indices of net types i for each of the $3m$ gates:

$$C_j = \{mB + 1\}; \quad O_j = \{1 + D_{j-1}, \dots, D_j\}; \quad N_j = C_j \cup O_j = \{1 + D_{j-1}, \dots, D_j\} \cup \{mB + 1\}$$

Finally, we set $s_i = 1 \quad \forall i$; $L = B + 1$ and $Z = \sum_l d_l + m$

Since the *3-PARTITION* problem is NP-complete for values of B that are bounded by a polynomial function of m , this transformation can be accomplished in polynomial time. We have created an instance of the *circuit-partitioning* problem in which each of the first mB nets is used by only one gate and the last net (net $mB + 1$)

is used by every gate. Thus, $|C_j|=1 \forall j$ and $|N_j| = d_j + 1 < B + 1 = L \forall j$.

Finally, we show that if we have a solution to this instance of the *circuit-partitioning* problem with an objective function value less than or equal to $Z = \sum_l d_l + 7m$, then we can construct a solution to the corresponding instance of the *3-PARTITION* problem. First, we show that in an optimal solution to the *circuit-partitioning* problem with this objective function, each chip must include exactly $L = B + 1$ nets. Suppose this is not so. Then there must exist some chip with B or fewer nets. Without loss of generality let one such chip be chip 1. Then, since net $mB + 1$ must be in every chip, the number of the first mB nets in chip 1 must be less than or equal to $B - 1$. Therefore, the number of the first mB nets that must be in the remaining chips must be greater than or equal to $mB - (B - 1) = (m - 1)B + 1$. Again, since net $mB + 1$ must be in every chip, there is only room for B other nets in each group. Thus, the remaining $(m - 1)B + 1$ nets can not be in only $m - 1$ chips. That is, they must be in at least m additional chips and the number of chips must therefore be at least $m + 1$. However, if this is so, each of the mB unique nets will be loaded exactly once and the common net (net $mB + 1$) must be loaded at least $m + 1$ times. Thus, the net packaging cost must be at least $\sum_l d_l + m + 1 > Z$. Therefore, each chip must contain exactly $L = B + 1$ nets.

For each (j, g) pair, we have $X_{ijg} = 1 \forall i \in N_j$. Also, $Z_{ig} = 1$ if $\exists j$ such that gate j is included in chip g (i.e., $Y_{jg} = 1$) and $i \in N_j$; and $Z_{ig} = 0$ otherwise. By construction, the set N_j consists of d_j unique nets (numbered $1 + D_{j-1}$ through D_j) plus net $mB + 1$. Thus, if gate j is included in chip g , we include index j in P_g . Since each gate j will be in only one chip and since the total number of unique nets in each chip is B , this assignment of indices j to sets P_g will result in $\sum_{j \in P_g} d_j = B$ for all sets P_g . Hence, the assignment will be a solution to the *3-PARTITION* problem. ■

APPENDIX C (BOUNDS ON THE NUMBER OF TIMES NET TYPE i MUST BE PACKED)

C.1 First Lower Bound: Let us introduce the following notation:

$$\delta(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{otherwise} \end{cases}$$

\mathcal{M}_i = the set of indices of gates j that use net type i or that are part of a megagate any one of whose constituent gates uses net type i .

With this notation, a lower bound on the total number of net type i usage is given by:

$$U_i = \sum_k \delta(\sum_{j \in \mathcal{M}_i} a_{kj}) \tag{10}$$

The inner summation equals the number of gates that use both net type k and net type i . The $\delta(\cdot)$ function sets this total to 1 if there is at least one such gate. Summing over all net types k , U_i is the total number of net types needed by all gates that require net type i . If this number is less than or equal to the chip capacity, L , then all of these gates could, at least from the perspective of net type i , be in the same chip. In that case, U_i indicates that net type i need only be packed once. More generally,

$$\left\lceil \frac{U_i}{L} \right\rceil \tag{11}$$

is a lower bound on the number of times net type i must be packed, where $\lceil x \rceil$ is the smallest integer greater than or equal to x .

C.2 Second Lower Bound: Let j_1 and j_2 be two gates such that $j_1, j_2 \in \mathcal{M}_i$. If, at some node in the branch and bound tree, gates j_1 and j_2 are forced to be in different chips, then net type i must be packed at least twice. We note that j_1 and j_2 may be forced to be in different chips implicitly. For example, j_1 may be forced to be in the same chip as some gate j_3 , j_2 may be forced to be in the same chip as gate j_4 and, gates j_3 and j_4 may be constrained to be in different chips. This would implicitly constrain j_1 and j_2 to be in different chips.

C.3 Third Lower Bound: Ersov and Kozuhin [9] showed that

$$\left\lceil \frac{n}{\lfloor t \rfloor} \left(1 - \frac{t - \lfloor t \rfloor}{1 + \lfloor t \rfloor} \right) \right\rceil \tag{12}$$

is a lower bound on the chromatic number of a graph, where $n = |V_i|$, $p = |E_i|$, $t = n - \frac{2p}{n}$ and $\lfloor x \rfloor$ is the largest integer less than or equal to x . Ersov and Kozuhin [9]

outlined the construction of graphs with n vertices and p edges that attain this lower bound. However, it should be noted that (12) is computationally intensive. This is important because the bound may need to be computed for each common net type i at each node of the branch and bound tree. In fact, the bound is likely to be computed multiple times for each common net type at each node in the tree.

To alleviate the computational cost which is involved in computing (12), we compute chromatic numbers of graphs G' that are constructed by deleting vertices and any edges incident on the deleted vertices from G . This relationship is worth noting because the lower bounds on G' using (12) may actually be **tighter** than this obtained for G . This is so because $\chi(G)$ is also an upper bound on $\chi(G')$. Thus, a lower bound on the chromatic number of G' is also a lower bound on the chromatic number of G .

To summarize, the process of successfully deleting the vertex (and incident edges) with the smallest degree coupled with the bound provided by (12) constitutes the third lower bound. The process is continued until a graph is constructed in which

its number of vertices n exceeds the best lower bound which has hitherto been computed. This process proved to be very effective for problems with small chip capacities, which were the most difficult problems to solve using our approach.

REFERENCES

1. Askin, R.G. and K. S. Chiu, "A Graph Partitioning Procedure for Machine Assignment and Cell Formation in Group Technology," *International Journal of Production Research*, Vol. 28, pp. 1555-1572, 1990.
2. Boothroyd, G., *Assembly Automation and Product Design*, Marcel Dekker, New York, 1992.
3. Chandrasekharan, M.P. and R. Rajagopalan, "An Ideal Seed Non-Hierarchical Clustering Algorithm for Cellular Manufacturing," *International Journal of Production Research*, Vol. 24, pp. 451-464, 1986a.
4. Chandrasekharan, M.P. and R. Rajagopalan, "MODROC: An Extension of Rank Order Clustering for Group Technology," *International Journal of Production Research*, Vol. 24, pp. 1221-1233, 1986b.
5. Chu, C.H., "Manufacturing cell formation by competitive learning," *European Journal of Operational Research*, Vol. 69 (3), pp. 292-311, 1993.
6. Chung, Y., and A. Kusiak, "Grouping Parts with a Neural Network," *Journal of Manufacturing Systems*, Vol. 13, pp. 262-275, 1994.
7. Ersov, A.P. and G.I. Kozuhin, "Estimates of the Chromatic Number of Connected Graphs," *Soviet Mathematics*, Translation of Doklady Akademii Nauk SSSR, Vol. 3, pp. 50-53, 1962.
8. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.
9. Harhalakis, G., Nagi, R. and J.M. Proth, "An Efficient Heuristic in Manufacturing Cell Formation for Group Technology Applications," *International Journal of Production Research*, Vol. 28, pp. 185-198, 1990.
10. Harhalakis, G., Proth, J.M. and X. L. Xie, "Manufacturing cell design using simulated annealing: an industrial application," *Journal of Intelligent Manufacturing*, pp. 185-191, 1990.
11. Kaparthi, S., Suresh, N.C. and R.P. Cerveny, "An improved neural network leader algorithm for part-machine grouping in group technology," *European Journal of Operational Research*, Vol. 69, pp.342-356, 1993.
12. King, J.R., "Machine-Component Group Formation in Group Technology," *OMEGA: The International Journal of Management Science*, Vol. 8, pp. 193-199, 1980a.
13. King, J.R., "Machine-Component Grouping in Production Flow Analysis: An Approach Using a Rank-Order Clustering Algorithm," *International Journal of Production Research*, Vol. 18, pp. 213-232, 1980b.
14. Kumar, K.R., Kusiak, A. and A. Vannelli, "Grouping of Parts and Components in Flexible Manufacturing Systems," *European Journal of Operational Research*, Vol. 24, pp. 387-397, 1986.
15. Kusiak, A., "The Generalized Group Technology Concept," *International Journal of Production Research*, Vol. 25, pp. 561-569, 1987.
16. Liu, C.M. and J.K. Wu, "Machine cell formation: using the simulated annealing algorithm," *International Journal Computer Integrated Manufacturing*, Vol. 6 (6), pp. 335-349, 1993.
17. Maimon, O. and A. Shtub, "Grouping Methods for Printed Circuit Boards Assembly," *International Journal of Production Research*, Vol. 29, pp. 1379-1390, 1991.
18. McGinnis, L. F., J.C. Ammons, M. Carlyle, Ranmer L., Depuy, G. W., Ellis, K. P., Tovey C. A. and H. Xu, "Automated Process Planning for Printed Circuit Card Assembly," *IIE Transactions*, 24, pp. 18- 26, 1992.
19. Papadimitriou, C.H. and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.
20. Pierreval, H. and M. F. Plaquin, "A Genetic Algorithm Approach to Group Machines into Manufacturing Cells," *Proceeding of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pp. 267-271, 1994.

21. Rajagopalan, R. and J.L. Batra, "Design of Cellular Production Systems: A Graph Theoretic Approach," *International Journal of Production Research*, Vol. 13, pp. 567-579, 1975.
22. Rockwell, T.H. and W.E. Wilhelm, "Material Flow Management in Cellular Configurations for Small-Lot Circuit Card Assembly," *International Journal of Production Research*, Vol. 28, pp. 573-594, 1990.
23. Shafer, S.M. and Meredith, J.R., "A Comparison of Selected Manufacturing Cell Formation Techniques," *International Journal of Production Research*, Vol. 28, pp. 661-673, 1990.
24. Shtub, A., "Modeling Group Technology Cell Formation as a Generalized Assignment Problem," *International Journal of Production Research*, Vol. 27, pp. 775-782, 1989.
25. Singh, N., "Design of cellular manufacturing systems: An invited review," *European Journal of Operational Research*, Vol. 69, pp. 284-291, 1993.
26. Vannelli, A. and K.R. Kumar, "A Method for Finding Minimal Bottle-Neck Cells for Grouping Part-Machine Families," *International Journal of Production Research*, Vol. 24, pp. 387-400, 1986.
27. Daskin, M., O. Maimon, A. Shtub and Braha D., "A Branch and Bound Algorithm for Grouping Components in Printed Circuits Board Production," *International Journal of Production Research*, to appear.
28. Lindsey, D., *The Design and Drafting of Printed Circuits*, revised ed., Bishop Graphics Inc., CA, 1984.
29. Breuer, M.A., "Min-Cut Placement," *Journal of Design Automation and Fault Tolerant Computing*, Vol. 1, pp. 343-362, 1977.
30. Fisher, D. H., "Knowledge Acquisition via Incremental Conceptual Clustering," *Machine Learning*, Vol. 2 (7), pp. 139-172, 1987.
31. Reich, Y., "Measuring the Value of Knowledge," *International Journal of Human-Computer Studies*, Vol. 42, pp. 3-30, 1995.

CHAPTER 12

PHYSICAL DESIGN OF PRINTED CIRCUIT BOARDS: GENETIC ALGORITHM APPROACH

In this chapter, we employ a genetic algorithm to search the space of alternative solutions. The fundamentals of this approach are outlined as they apply to the circuit-partitioning problem which was presented in Chapter 11. Computational results are provided for this approach and are compared with the heuristic solution approach that was presented in Chapter 11 (Section 11.4).

12.1 INTRODUCTION

In Chapters 2 and 6, we established the hypothesis that view the act of design as an *evolutionary process* and the design itself at *any* stage of its development as a *tentative solution* to the original requirements. We may also compare this process to *Darwinian natural selection*, for both non-adapted organisms and incorrect designs are weeded out, leaving the stronger to continue in the competition.

The hallmarks of Darwinian evolution can be summarized through an example. At any given time there is a population of mice. Some of them are faster than other mice. These faster mice are less likely to be eaten by cats, and therefore more of them survive to do what mice do best: make more mice. Of course, some of the slower mice will survive just because they are lucky. This surviving populations of mice starts breeding. The breeding results in a good mixture of mice genetic material: some slow mice breed with fast mice, some fast with fast, and some slow with slow. And on the top of that, nature mutates every once in a while some of the mouse genetic material. The resulting offspring of the mice will (on average) be faster than these in the original population because more faster parents survived the cats.

The above process is termed *natural selection*. By this process, organisms are constantly *tested* against the environment and those that are genetically endowed to survive may be said to *fit* relative to that environment. If the environment changes then some forms of the organisms within the population may become fit relative to the new environment while other forms may die out. Thus, organisms appear to constantly *adapt* to its surroundings. We used these concepts of *testing* and *adaptation* as underlying the design process in general.

During the last three decades there has been a growing interest in genetic

algorithms, which rely on analogies with Darwinian natural selection. Genetic Algorithms (GAs), which use a vocabulary borrowed from natural genetics, have been quite successfully applied to optimization problems like wire routing, scheduling, adaptive control, cognitive modeling, transportation problems, traveling salesman problems, etc. [1]. A GA follows a step-by-step procedure that closely matches the story of the mice. A GA performs a multidirectional search by maintaining a population of potential solutions and encourages information formation and exchange between these directions. The population undergoes a simulated evolution: at each generation the relatively “good” solutions reproduce, while the relatively “bad” solutions die. To distinguish between different solutions, we use an objective (evaluation) function which plays the role of an environment.

The purpose of this chapter is a demonstration of the applicability of genetic algorithms as suggested in [2] (for similar group technology production problems) to the partitioning phase of electronic circuit design. The circuit-partitioning problem is addressed and formulated in Chapter 11. Thus, regardless of the claim that the evolutionary model in the specific Darwinian sense is a universal feature of design processes, it has a heuristic value in carrying out the act of engineering design.

The remainder of the chapter is organized as follows. In Section 12.2, we briefly outline the key elements of a genetic algorithm. In Section 12.3, we present a genetic algorithm approach to solving the circuit-partitioning problem that was presented in Chapter 11. Computational results obtained from using the genetic algorithm are summarized in Section 12.4. In Section 12.5, we show how catalogue search for fixed configuration solutions can be solved with a genetic-search approach. Section 12.6 concludes the chapter.

12.2 THE GENETIC ALGORITHM APPROACH

Traditional heuristic algorithms construct a single solution and then attempt to improve on that solution using a variety of rules. Both the construction algorithms and the improvement procedures are often complex. This complexity is justified since only a single solution is being manipulated and the procedures must attempt to avoid being stuck at a local optimum. Genetic algorithms (GAs), by contrast, simultaneously maintain a large *population* of solutions and tend to manipulate each solution in much simpler ways [1]. This section briefly outlines the key elements of a genetic algorithm. The description focuses first on how one might develop a genetic algorithm for a facility design problem (see Chapter 13), simply because the description of a genetic algorithm in this context is simpler than that in a circuit-partitioning context. Following the introduction to genetic algorithms, the discussion focuses on the development of a genetic algorithm for the circuit-partitioning problem.

To develop a genetic algorithm, schemes for *encoding*, *decoding*, and *evaluating* a solution are required. Often solutions are encoded as bit streams. Thus, for example, in a facility design problem with N candidate parameters (each fixed to one of two possible levels), one natural encoding scheme would represent each candidate

parameter-level as a bit in an N -bit word. Setting the j^{th} bit to 1 would indicate that the j^{th} parameter is to be assigned to its second level; a 0 would indicate that the j^{th} parameter is to be assigned to its first level. Once the scheme is decoded, it must be evaluated. In many facility design contexts, the output performance measure is evaluated by means of the execution of a computer simulation. Often the encoding, decoding and evaluation decisions are intimately linked.

Genetic algorithms begin by randomly generating a population of candidate solutions. Given such a population, a genetic algorithm generates a new candidate solution (population element) by selecting two of the candidate solutions as the *parent* solutions. This process is termed *reproduction*. Generally, parents are selected randomly from the population with a bias toward the better candidate solutions. Given two parents, one or more new solutions are generated by taking some characteristics of the solution from the first parent (the father) and some from the second parent (the mother). This process is termed *crossover*. For example, in the facility design context, we might randomly select a crossover bit location, n ($1 < n < N$). Two child solutions could then be generated. The first child would inherit the first n siting characteristics from the father and the remaining $N-n$ characteristics from the mother. The second child would inherit the first n from the mother and the remaining $N-n$ from the father solution. This is illustrated in Figure 12.1 for a problem with $N=6$ and $n=2$. Finally, once child solutions are generated, genetic algorithms allow characteristics of the solutions to be changed randomly in a process known as *mutation*. In the binary encoding representation, typically, with a small probability (e.g., 0.01) each bit position is changed from its current value to the opposite value. With this probability of a single bit changing, the probability of a child solution remaining unchanged is $(.99)^6 = .9415$. With probability greater than .88, both child solutions would remain unchanged.

Once candidate child solutions have been generated, they are decoded and evaluated. Typically, if a child solution is better than the worst solution in the parent population, the child solution replaces the worst element in the parent population. The process continues until some *termination criterion* is satisfied. Typical termination criteria include stopping when (1) the value of the best and worst elements in the parent population are either identical or sufficiently close to each other; (2) the value of the *best* solution in the parent population has not improved after N_1 successive children have been generated; (3) the value of the *average* solution in the parent population has not improved after N_2 successive children have been generated; or (4) N_3 children have been generated. Goldberg [1] provides an excellent introduction to genetic algorithms along with theoretical results suggesting why such relatively simple algorithms should work well.

PARENT SOLUTIONS:	Father <i>(1,0,1,1,0,1)</i>	Mother (0,1,0,0,1,1)
RANDOMLY SELECTED CROSSOVER POINT:	Between bits 2 and 3	
CHILD SOLUTIONS:	<i>(1,0,0,0,1,1)</i>	(0,1,1,1,0,1)
<i>(Father Characteristics shown in italics to indicate more clearly the inheritance process)</i>		

Figure 12.1 Example Crossover Operation in a Facility Design Context

12.3 A GENETIC ALGORITHM FOR THE CIRCUIT-PARTITIONING PROBLEM

In the context of circuit-partitioning, encoding, decoding, and evaluating solutions are somehow trickier processes than in the facility design context outlined above. Similarly, the crossover and mutation operations need careful specification. As noted above, decisions regarding encoding, decoding and evaluating solutions must often be linked. In the circuit-partitioning problem formulated in Chapter 11, each gate is included in exactly one chip (package). Thus, one natural encoding scheme is to represent a solution as a *permutation* of the indices of the gates. The decoding scheme would then process the gates in the order in which they appear in the permutation, assigning each gate to a chip in some manner. Once the chips are formed, the solution can be evaluated simply by summing the costs of the nets required in each chip over all chips.

Three alternate decoding schemes are developed and tested. In all schemes, nets are included in chips based on the order in which the gates appeared in the permutation. In the first approach, chips are formed *sequentially*. If the next gate in the permutation can be included in the chip under consideration (the number of additional nets required by the gate that is packed in the chip is less than or equal to the number of additional nets that could be included in the chip), the gate is included in the chip; if not, a new chip is initiated. In the second decoding scheme, for each gate, the emerging chips are tested, beginning with the first chip constructed, until a chip is found in which the gate can be included without violating the constraint on the maximum number of nets in a chip. The gate is included in the *first* chip in which it can feasibly be included. If no such chip exists, a new chip is initiated. In the third decoding approach, each gate is included in the *best* of the emerging chips, where 'best' is defined as the chip which already includes the most nets required by the gate under consideration, and which can feasibly accommodate the additional nets required by the gate. Again, if no such chip exists, a new chip is initiated. Note that

each of these decoding schemes, no matter how the gates are assigned to chips, differs from the heuristic outlined in Chapter 11 (Section 11.4). In that heuristic, *chips are constructed sequentially* and a search is conducted over all unassigned gates for the best to include in the emerging chip. In the decoding schemes used in the genetic algorithms, *gates are processed sequentially* based on their order of appearance in the permutation.

With solutions represented as permutations of the integers 1 to *N*, the crossover process must be carefully specified to ensure that the child solutions are also permutations. Goldberg [1] summarizes three crossover processes that satisfy this need. In the genetic algorithm developed for the circuit-partitioning problem, the *partially matched crossover* technique is used. In this approach, two crossover points are identified. The first child is constructed as follows. First, its permutation is set equal to that of the father. Next, the portion of the permutation between the two crossover points is replaced by the same portion of the mother’s permutation element by element. As each element is replaced, the previous element in the child at that location is moved to the location in the child of the value of the replacing element. This is illustrated in Figure 12.2 for parents characterized by a permutation of the integers from 1 to 9. Note that both children are also permutation of the integers from 1 to 9.

PARENT SOLUTIONS:	Father	Mother
	(9,8,4,6,7,5,2,1,3)	(2,4,5,1,3,8,6,9,7)
TWO RANDOMLY SELECTED CROSSOVER POINTS:	Between positions 2 and 3	
	Between positions 5 and 6	
PARENT SOLUTIONS WITH CROSSOVER DELIMITERS:	Father	Mother
	(9,8 4,6,7 5,2,1,3)	(2,4 5,1,3 8,6,9,7)
CONSTRUCTION OF FIRST CHILD SOLUTION:		
1. Copy of Father:	(9,8 4,6,7 5,2,1,3)	
2. Replace 1 st Element:	(9,8 5,6,7 4,2,1,3) <i>5 replaces 4;</i> <i>4 moves to position previously occupied by 5</i>	
3. Replace 2 nd Element:	(9,8 5,1,7 4,2,6,3) <i>1 replaces 6;</i> <i>6 moves to position previously occupied by 1</i>	
4. Replace 3 rd Element:	(9,8 5,1,3 4,2,6,7) <i>3 replaces 7;</i> <i>7 moves to position previously occupied by 3</i>	
SECOND CHILD:	(2,5 4,6,7 8,1,9,3)	
<i>(Father Characteristics shown in italics to indicate more clearly the inheritance process)</i>		

Figure 12.2 Example Generation of A Single Child Using The Partially Matched Crossover Technique

Finally, the mutation process is changed. Instead of examining each *bit* and changing it with a specified permutation probability, we elect to mutate a solution with a given permutation probability (e.g., 0.2). If a solution is elected to undergo mutation, we randomly select two positions and exchange the elements in the permutation in these positions.

12.4 COMPUTATIONAL RESULTS

The four data sets (DATA1, DATA3, CLUSTRA and CLUSTAX), which were given in Chapter 11, were used in testing the genetic algorithm. Table 11.1 is a summary of key features of the four data sets. Finally, Table 12.1 summarizes the key input parameters for the genetic algorithm. The size of the parent population differs for the four data sets. The genetic algorithm is terminated when either (1) the values of the best and worst solutions in the parent population are equal; or (2) a pre-specified limit on the number of children to be generated is reached. The magnitude of this limit differs for each of the four data sets used in the tests of the algorithm. This, however, was the cause of termination in only 1 of the 181 genetic runs. The probability of mutating a child solution is 0.2 for all data sets.

Table 12.1 Input Parameters for the Genetic Algorithm

	DATA SET			
	<u>DATA1</u>	<u>DATA3</u>	<u>CLUSTRA</u>	<u>CLUSTAX</u>
Genetic Algorithm Parameters:				
Population Size	100	500	500	500
Maximum Number of Children	500	25,000	50,000	50,000
Mutation Probability	0.2	0.2	0.2	0.2

Table 12.2 compares the 3 variants of the genetic algorithm (based on the 3 decoding algorithm outlined above) in terms of the ability of the algorithms to find the optimal solution for the 4 data sets. Decoding using the BEST algorithm and the SEQUENTIAL algorithms resulted in about the same average likelihood of finding the optimal solution. Using the FIRST algorithm resulted in slightly poorer performance. Overall, the genetic algorithm found the optimal solution in almost 80 percent of the runs. This percentage decreased with the problem size. For the largest problem, the genetic algorithm found the optimal solution in about 53 percent of the runs. It is likely that this percentage could be increased by running the genetic algorithm multiple times with different random number seeds. If a looser termination criterion were employed, the execution time of the genetic algorithm might not be increased dramatically.

Table 12.3 shows the average execution time for the three decoding algorithms and the four data sets. As expected, the execution time for each algorithm increases with problem size. The average time for the SEQUENTIAL algorithm exceeded that

of the other two algorithms for all data sets. This is so despite the fact that the SEQUENTIAL algorithm takes less time to evaluate each child. However, as shown in Table 12.4, the SEQUENTIAL algorithm requires considerably more children to be generated before the values of the best and worst solutions in the parent population are equal.

Balancing the solution quality (Table 12.2), execution time (Table 12.3) and the number of required children (Table 12.4), it seems as though decoding the permutations based on putting the next gate into the BEST emerging chip is the most cost effective approach.

Table 12.2 Percent of Runs in which Genetic Algorithm Found Optimal Solution

DATA SET:	DECODING ALGORITHM			
	<u>BEST</u>	<u>FIRST</u>	<u>SEQ'L</u>	<u>AVERAGE</u>
DATA1	100.00	100.00	100.00	100.00
DATA3	97.62	92.86	95.24	95.24
CLUSTRA	96.67	96.67	86.67	93.34
CLUSTAX	52.86	48.57	57.14	52.86
WEIGHTED AVERAGE	80.66	77.90	80.11	79.56

Table 12.3 Average Solution Time for Genetic Algorithm

DATA SET:	DECODING ALGORITHM			
	<u>BEST</u>	<u>FIRST</u>	<u>SEQ'L</u>	<u>AVERAGE</u>
DATA1	1.52	1.52	1.75	1.6
DATA3	28.42	29.07	33.36	30.28
CLUSTRA	46.37	46.04	51.93	48.11
CLUSTAX	109.53	100.54	120.40	110.16
WEIGHTED AVERAGE	57.28	53.88	63.64	58.27

Table 12.4 Average Number of Children in Genetic Algorithm

DATA SET:	DECODING ALGORITHM			
	<u>BEST</u>	<u>FIRST</u>	<u>SEQ'L</u>	<u>AVERAGE</u>
DATA1	152	160	226	179
DATA3	2,169	2,383	3,367	2,640
CLUSTRA	2,506	2,774	3,809	3,030
CLUSTAX	2,940	3,010	4,556	3,502
WEIGHTED AVERAGE	2,100	2,224	3,241	2,522

Finally, Figures 12.3 and 12.4 compare the heuristic that was presented in Chapter 11 (Section 11.4) with the three decoding schemes for the genetic algorithm

in terms of the average (Figure 12.3) and maximum (Figure 12.4) percentage error in the total net costs (the cost incurred when nets are included in chips) when compared with the optimal net costs. Even in the largest of the problems, the average percentage error for the worst of the genetic algorithms (the SEQUENTIAL decoding scheme) was under 1 percent. The heuristic of Chapter 11 gave an average error of almost 6 percent for this data set. Similar results were found for the maximum percentage error. The heuristic of Chapter 11 resulted in 25 percent errors for the largest data set, while the worst of the genetic algorithm results was within 5 percent of the optimal solution. The magnitude of both the average and maximum percent errors tends to grow with the size of the problem for both the heuristic of Chapter 11 and the genetic algorithm with any of the decoding schemes.

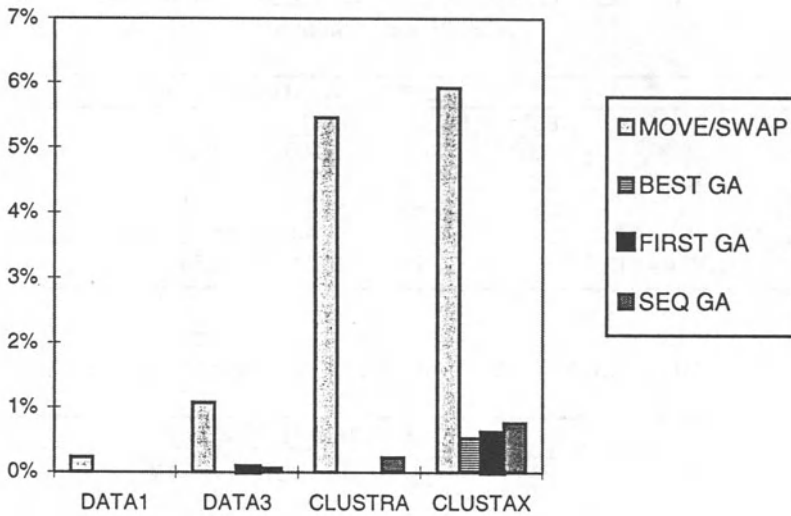


Figure 12.3 Average Percent Error in Heuristic Solutions

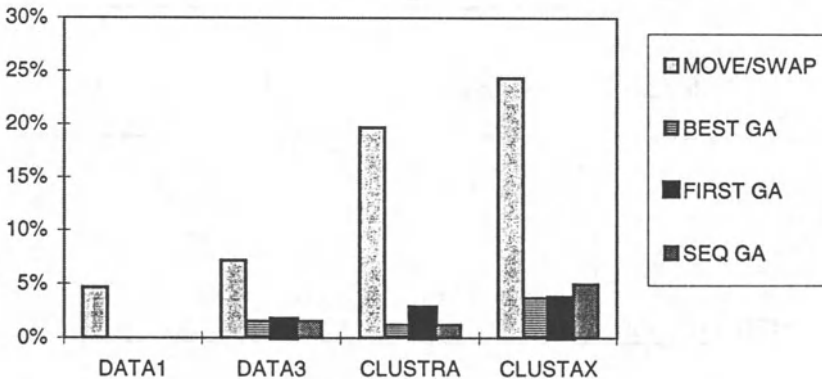


Figure 12.4 Maximum Percent Error in Heuristic Solutions

12.5 OTHER APPLICATIONS OF GENETIC ALGORITHM

12.5.1 THE CATALOGUE SELECTION PROBLEM

Genetic algorithms can also be developed for solving fixed configuration tasks. In other words, the types of parts in the assembly and how they are connected are specified by the designer. The remaining task is to choose which particular part numbers (or types) to use. The designer specifies what type of parts are in the configuration and how they are connected. Configuring a design from parts out of a catalogue, which satisfies predetermined goals and constraints, requires searching large numbers of combinations of parts. We refer to this problem as the *catalogue-selection* problem. For example, Figure 12.5 shows a hoist that a designer might need to specify. The motor, worm and wormgear, shaft set parts and bearings must be chosen from those listed in a catalogue. The engineer must satisfy externally imposed constraints such as power and speed and internal compatibility constraints between adjacent parts (such as Stress Limit, Lifting Speed, and Thrust Force). All of these is to be done while minimizing an objective such as production cost. There are typically a large number of possible solutions from which to choose, which makes exhaustive search for the best solution difficult.

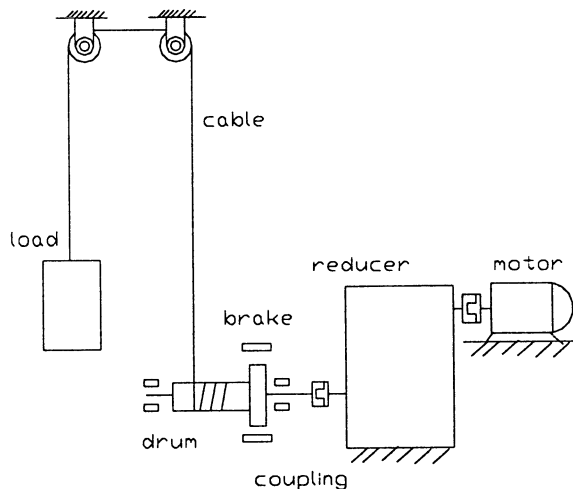


Figure 12.5 The Working Principle of A Hoist

As another example of the *catalogue-selection* problem, consider a situation in which the designer addresses the problem of selecting the appropriate levels of the various attributes to be engineered into the product. For example, in designing a flexible manufacturing system a designer would be interested in determining for the attributes *raw material type* and *material handling policy*, the specific level of each, *steel* or *aluminum* (for raw material type), and, *first come first served* or *high value*

first (for material handling policy) to employ in the proposed new system.

Optimizing search techniques are not practical for searching for parts in catalogues because the computation time is often excessive. Rule-based approaches suffer from being domain specific and thus are not appropriate as general solution methods. Thus, a feasible alternative for solving configuration tasks is to use a technique based on genetic search.

12.5.2 OUTLINE OF THE GENETIC ALGORITHM

One important question is how to encode products so that they could be searched with GA. Based on the algebraic representation of design artifacts, a product (configuration, e.g., wormgear reducer) is described in terms of part types (a group of objects which are similar but have different sizes, e.g., stepper motor). Every part type can be further described by a set of attributes (a feature of a part type that other part types can connect to or get or receive information through, e.g., axial pitch). Each attribute can be described by its dimension. A relation states that two attributes are in contact. A candidate solution is a collection of part types with connections specified between their attributes. For a problem to be solved within the GA paradigm, one needs to formulate the problem using a string structure. In the *catalogue-selection* problem, we consider a chromosome (i.e. string) to represent a product. The parts of the product would correspond to *genes* and the levels, i.e., types the part could take, to *alleles*.

For purposes of GA, the products are represented by binary strings. For instance, 11111-01110-01101-11011-11010, represents load-31 connected to drum-14 connected to wormgear-13 connected to worm-27 which is connected to motor-23. Each part is further specified by a set of attributes and their respective values. For example, wormgear-13 is described in terms of the Wormgear Teeth Number $N_g = 50$, Wormgear Face Width $F_g = 0.4941(\text{in})$, Wormgear Pitch Diameter $D_g = 2.548(\text{in})$, and Wormgear Outside Diameter $D_{g0} = 2.648(\text{in})$.

Each string (i.e., product) has a fitness value which rates how well the candidate achieves the objective function (e.g., production cost), and how much it deviates from the constraints between the attribute values (dimensions) of adjacent parts.

An interesting aspect of the approach introduced above is that parts are represented only as part numbers. When a catalogue of parts is not available, the part type attributes have to be determined from scratch. In this case, a genetic algorithm can be used to solve a non-linear programming optimization problem. The representation of a candidate solution takes into account the attributes of the part types (e.g. Wormgear Teeth Number, Wormgear Face Width, Wormgear Pitch Diameter, Wormgear Outside Diameter etc.) The binary representation of candidate solutions should code a string of *real* attribute values (thus, exploring a continuous domain of solutions). The fitness function is similar to the one used for the *catalogue-selection* problem.

12.6 Summary

In this chapter, we presented an approach for solving the circuit-partitioning problem using a genetic algorithm. Solutions were coded as a permutation of the gate index. Three decoding algorithms were outlined. The results suggest that the genetic algorithm can find the optimal solution in a large percentage of the problems, particularly for the smaller data sets. The genetic algorithm results were significantly better than the sequential chip construction heuristic of Chapter 11 in terms of both the average and maximum percentage deviations from the optimal value. Naturally, the genetic algorithms took considerably longer to execute. The use of genetic algorithms in the context of physical design of printed circuit boards seems to be a promising area for future research.

This chapter has also shown how to solve fixed configuration problems with genetic search. Since genetic search is much faster than optimization-based methods, the designer may use it to explore larger design spaces that were previously possible.

REFERENCES

1. Goldberg, D. E., *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley, New York, 1989.
2. Daskin, M. S., "An Overview of Recent Research on Assigning Products to Groups for Group Technology Production Problems," *Israeli Institute of Industrial Engineers Conference, Tel-Aviv*, 1991.

CHAPTER 13

ADAPTIVE LEARNING FOR SUCCESSFUL DESIGN

In this chapter we want to evaluate which of many parameters (each is set at various possible levels) composing a design solution have the greatest likelihood of satisfying a given set of functional requirements. The design's functional requirements are represented by a set of prespecified limits that determine where the output responses should fall. Adopting the probabilistic paradigm presented in Chapter 7 and the methodology provided in Chapter 8 for quantifying how well a proposed design satisfies the governing requirements (in probabilistic terms), we present a method for adaptive learning of successful designs that is based on the use of statistical experimental design and a stochastic search algorithm. In Chapter 19, we present a real industrial problem of designing a flexible manufacturing system that is solved based on the proposed algorithm.

13.1 INTRODUCTION

13.1.1 MANAGING THE INTRICATE CORRESPONDENCE BETWEEN FUNCTION AND STRUCTURE

As described in Chapter 5, the primary concept of FDT is that the design process is a mapping (synthesis) of the desired set of specifications (describing the desired functions and constraints of the final product) onto the artifact description (the final detailed product description). Starting with the stated functional requirements, the designer generates a solution by defining design parameters that best satisfy the functional requirements and describe the solution's physical layout. Functional requirements refer to the set of specific requirements that completely describe the perceived needs associated with the output performance measures of the design solution. Often the output performance measures have noise factors associated with them. Noise factors are those variables in a design that are uncontrollable or unpredictable, and represent an uncertainty that the desired response will achieve a desired value. In experimental-design terminology [1], the input parameters and structural assumptions composing a design are called *factors*, and the output performance measures are often termed *responses*.

This chapter is based on the following characteristics. First, we deal with situations in which the rule of direct correspondence between the output performance measures and the design parameters is highly complex due to coupling effects (interactions) that preclude any possibility of an analytical solution. In these design scenarios the designer wants to find out which of possibly many parameters (each fixed to one of various possible levels) composing a design solution, have the greatest effect on one or more output performance measures. Performance measures have functional requirements (design goals) associated with them which are represented by a set of requisite tolerances. Second, we assume that the preliminary structure of the design is already determined at this phase. In particular, the set of design parameters and structural assumptions are considered fixed aspects of the design problem. The design problem is related to determining the optimal combination of the various parameter levels.

In this chapter, the output performance measures are evaluated by applying a computer simulation model. For example, simulation is often used as a technique for evaluating performance measures in flexible manufacturing system (FMS) design (see Chapter 3). In FMS, possible factors include: number of machines, queue discipline, buffer sizes, conveyor speeds, and machine groupings into cells. Examples of responses are: parts throughput, time in system, machine utilization, and profitability. FMS is a good example of a complex system. If a change is made at a particular work station, its impact on the performance of the overall system may not be predictable by simple formal analysis, which results in the use of a computer simulation model.

Without a numerical basis for comparison, however, the final selection of a design solution involving many functional requirements can only be made on a subjective basis. To resolve this difficulty, we provided in Chapter 8 a rational means for quantifying how well a proposed design satisfies the governing requirements in terms of its overall *probability* of successfully achieving the functional requirements. We employ this methodology in this chapter.

This chapter provides a methodology for adaptive learning of successful designs (termed as P-learning) that is based on the use of statistical experimental design and optimization techniques. As mentioned earlier, without any loss of generality, we define "experiment" as the execution of a computer simulation model. The P-learning algorithm constructs a sequence of samples (populations of candidate solutions), where each sample includes particular designs to simulate. Given such a sample, the P-learning algorithm learns more about the design's behavior. In particular, which factors and factor interactions appear to satisfy the governing requirements in terms of the overall success probability. As the desired information is obtained, the P-learning algorithm generates new candidate solutions (sample elements) with a bias toward candidate solutions that include better parameter levels (in terms of their overall success probability). This adaptive mechanism is used to approximate the *smallest probable set*, which is the smallest set that includes all the most probable solutions (in terms of being included in a successful design, if it exists). This strategy is justified in Chapter 7 (Section 7.4) by showing that under certain assumptions the smallest high probability set (which is termed as the *typical set*) is a fairly small set

relative to the number of potential designs. Once candidate solutions have been generated, they are simulated. This process repeats itself until a certain termination criterion is satisfied. Typical termination criteria include stopping when the whole number of executed simulation runs exceeds a predetermined constraint.

13.1.2 THE APPLICABILITY OF THE METHODOLOGY

As mentioned in Chapter 2, a typical design paradigm can be generalized as a top-down or hierarchical progression involving three stages: conceptual design, preliminary design, and detailed design. There is an incremental refinement of the design description as the level of detail evolves from abstract concepts to drawings that support the finalization of the design product. The methodology proposed in this chapter better supports the preliminary and detailed design stages since it is assumed that the parameters and structural assumptions are considered fixed aspects of a pre-developed conceptual solution.

A further aspect of the proposed methodology -- of enhanced control over simulation experiments and specified levels -- supports incremental redesign activities in a global competitive marketplace. We are part of a marketplace that is becoming more global and more competitive every day. Incremental design has become the standard approach towards design in many areas. Most new products are only slightly modified from their predecessors with slight cosmetic or feature enhancements. In order to decide which new products to develop, large consumer goods companies often create several different prototypes, test market all of them, and develop whichever one sells the best. In this ever-changing environment, fast time-to-market has become critically important. Companies cannot be required to completely redesign their products. It is likely that they would have to modify their existing products by adding new features, modifying the specifications, or incorporating new materials and new technologies. If most of the design specifications do not change, computer companies cannot afford to redesign their products from scratch just because a new CPU is introduced. Likewise, in designing a new computer keyboard, many design issues have for the most part already been decided including which keys to include and in what order to place them.

Finally, in addition to evaluating design performance relative to a particular functional requirement, some type of multi-objective decision criteria is needed to select the optimal solution when there is more than one requirement to be satisfied. Traditionally, multi-objective decision functions involve a combination of scale factors for putting all objectives into equivalent units and weighing factors for describing the relative importance of each objective. In contrast, the use of success probability for optimizing design decisions is appealing because it offers a uniform and unbiased decision criterion for resolving multi-objective design problems. The use of success probability provides a method by which different functional requirements are integrated and normalized to units of probability.

The chapter is organized as follows. The problem formulation is provided in Section 13.2, followed by its detailed solution in Section 13.3. The solution approach

is illustrated in Section 13.4. Section 13.5 suggests a method for capturing the history of previous design processes. Section 13.6 summarizes the chapter.

13.2 PROBLEM FORMULATION

As mentioned earlier, this chapter deals with a situation in which the designer wants to find out which of possibly many parameters (each fixed to one of various levels) composing a design solution have the greatest effect on one or more output performance measures (responses). A particular output response is said to satisfy the corresponding functional requirement if it falls within the tolerance that represents the respective functional requirement. How well a proposed design satisfies the governing requirements is evaluated in terms of the probability of its successfully achieving these requirements.

As explained above, we deal with situations in which the rule of direct correspondence between the output performance measures and the design parameters is highly complex, precluding any possibility of an analytical solution. In this case, the output performance measures are evaluated by executing a computer simulation. Since the execution of a large simulation requires considerable computational effort, the problem formulation as well as the solution strategy incorporate computational constraints in terms of the number of simulation runs required.

The following notation will be used:

- I : The parameter index set, $I = \{1, 2, \dots, N\}$;
- J : The level index set, where we consider the case in which all parameters have the same number of levels M ;
- d : A feasible design solution $d = \left\{ \lambda_i^j \mid i=1,2,\dots,N, j \in J \right\}$, where $\lambda_i^j \in I \times J$, $|d| = N$, and exactly a single level $j \in J$ is assigned to each parameter $i \in I$;
- Ω : the design space $\Omega = \{d_1, d_2, \dots, d_q, \dots, d_{MN}\}$, representing the set of all feasible designs. In order to generate a solution, an extensive search of the design space might be required;
- $f: \Omega \rightarrow \mathfrak{R}^L$: a mapping from the design space to \mathfrak{R}^L . $f(d_q)$ denotes a vector of L output performance measures (also called responses) of the q th design. We denote r_l^q as the l th response of the q th design;
- $t_l = (\alpha_l, \beta_l)$: the l th design specification that is expressed in terms of a tolerance (also termed a design range) associated with the l th performance measure. α_l and β_l are the lower and upper limits of the tolerance, respectively. Let $T = \{t_l \mid l=1, 2, \dots, L\}$ denotes the desired set of specifications (describing the desired design range of the final product) associated with the L output performance

measures;

Given a candidate design solution d_q , the probability of successfully achieving the l th design specification is $P_{l_i}^q = \text{Prob}[\alpha_l \leq r_l^q \leq \beta_l]$. Therefore the probability of successfully achieving the set of independent specifications in T can be written as $P_{d_q} = \prod_{l=1}^L P_{l_i}^q$. The success probability P_{d_q} is evaluated in this chapter by the execution of a computer simulation. In order to generate the optimal solution, extensive simulation runs of the design space might be required. However, it is easy to imagine a complicated design problem in which we might be interested in dozens or even hundreds of different parameters. Thus, the number of simulation runs needed may prove quite inefficient. This computational constraint is tacitly assumed by the solution approach presented in later sections. Specifically, we associate a unit cost to the execution of a single computer simulation run, and require to find an optimal design by performing at most C computer simulation runs. The designer's task is to seek optimal combinations of parameter levels d_q^* (constituting the "best" design solution) that maximize the probability of successfully achieving the set of specifications T by performing at most C computer simulation runs. Subsequent to obtaining the optimal solution d_q^* , the designer may decide to examine the admissibility of the optimal solution by comparing $P_{d_q^*}$ with a threshold probability P_0 . In case $P_{d_q^*} < P_0$, a more elaborate model of the system with a larger number of parameters and levels must be considered. In general, the theoretical distribution of the output performance measure is unknown. Therefore, we develop procedures to estimate the parameters of the probability distribution that was computed from the observations obtained by executing the computer simulation.

13.3 ADAPTIVE LEARNING OF SUCCESSFUL DESIGN

13.3.1 THE PROBABILISTIC NATURE OF THE DESIGN PROCESS

The nature of the information involved in the search for a design solution may be either deterministic by showing which designs are categorically inferior, or probabilistic by identifying those designs which have the greatest probability of satisfying a given set of requirements. The methodology and the underlying P-learning algorithm presented in this chapter is founded on the probabilistic paradigm.

Recalling Chapter 8, the success probability of a design (i.e. combinations of factors levels) that relates to the satisfaction of a particular task can be computed as follows. Figure 13.1 illustrates three possibilities of a non-uniform probability distribution associated with a particular performance measure l of the design product

d. The mean and variance of each probability distribution are indicated by μ and σ^2 , respectively. The functional requirement of a product is represented by a tolerance $t_l = (\alpha_l, \beta_l)$. The probability of satisfying the functional requirement is depicted by the white area, denoted by $P_{t_l}^q = \text{Prob}[\alpha_l \leq r_l^q \leq \beta_l]$, which falls between the limits defined by the requisite tolerance. As shown in Figure 13.1, the success probability can be increased either by moving first the mean μ toward the desired response (denoted by the dashed line) or by reducing the variance σ^2 . In case some type of multi-objective decision criteria is needed to select the optimal solution, it is rational to quantify how well a proposed design satisfies the design goals in terms of the overall probability of successfully achieving a set of independent functional requirements, i.e. $P_{d_q} = \prod_{l=1}^L P_{t_l}^q$.

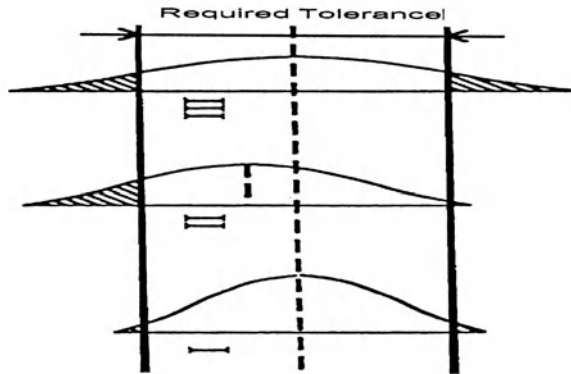


Figure 13.1 Probability of Satisfying A Functional Requirement

Suppose that there are M levels for each parameter and the designer wants to estimate how a particular parameter-level satisfies the governing requirements. One way to measure the effect of a particular parameter-level is to fix the parameter at the particular level, letting all other parameters vary over some set of values, and make simulation runs for each combination of parameter levels. Then to estimate the value of the actual mean μ and the variance σ^2 of the output response (in place of their precise values), which are used to compute the success probability as shown earlier (and as detailed in Appendix A). The main goal of the P-learning algorithm is to seek parameters levels that occur in successful designs.

Having explained the basic concepts of computing the probability of success; an outline of the P-learning algorithm follows. In the first stage of the P-learning algorithm, the designer has no information (*zero information* state) about which parameter levels satisfy the governing requirements. In the *zero information* state, the designer sets the initial representative sample, Ω^1 , to be an orthogonal array, for which each parameter-level appears with the same frequency. Given such a sample,

the P-algorithm learns more about the design's behavior in terms of which parameter levels appear to satisfy the governing requirements. As the desired information is obtained, the P-learning algorithm constructs a new sample of candidate solutions. Each new sample element is generated with a bias toward candidate solutions that include better parameter levels as determined by the experimental success probabilities. Once a new sample of candidate solutions, Ω^S , has been generated, each candidate solution is simulated. In the revised sample, each parameter-level appears with a frequency that depends on its success probability as computed by the previous sample. This process repeats itself until the total number of executed simulation runs exceeds a predetermined limit. In general, the P-learning algorithm manipulates the population of candidate designs in such a way that poor designs fade away and successful designs continually evolve (reminiscent of some genetic-based algorithms). If, however, the design problem involves dozens or even hundreds of different parameters, the experimentation becomes excessively time-consuming and impractical. In such a case, to screen out unimportant parameters, it might be *advisable* to apply (for the first orthogonal array) analysis of variance (ANOVA) and pooling up techniques at the beginning of the P-learning algorithm. A detailed description of the P-learning algorithm (including the initial screening procedure) is provided in the next two sections.

13.3.2 PRELIMINARIES

For ease of exposition, we consider the case of evaluating the design performance relative to a single functional requirement. When there is more than one criteria to be satisfied, the procedure presented in Section 13.3.2 can be applied. The following terminology will facilitate the description of the solution approach as presented in Section 13.3.3.

- r^q : the design response of the q th design evaluated by the function $f(d_q)$. For a multi-objective decision criteria, r^q denotes a vector of L output performance measures.
- ω : the number of independent replications at each parameter combinations (a design solution).
- S_i^j : the set of designs in Ω^S , each of which has the j th level assigned to its i th parameter.
- $\hat{\mu}_{\lambda_i^j} = \frac{\sum_{d_q \in S_i^j} r^q}{|S_i^j| \cdot \omega}$: the estimated mean response of all designs in S_i^j .

- $\hat{\sigma}_{\lambda_i^j}^2 = \frac{\sum_{d_q \in S_i^j} (r^q - \hat{\mu}_{S_i^j})^2}{(|S_i^j| \cdot \omega) - 1}$: the estimated variance of all designs in S_i^j . In

many cases [1] the mean residual error (*MSe*), as obtained from the ANOVA table that is constructed for the first orthogonal array, is used as an estimate of $\hat{\sigma}_{\lambda_i^j}^2$.

- $\hat{\mu}_{d_q}$: the estimated mean response of the design d_q . If $d_q \in \Omega^s$, $\hat{\mu}_{d_q}$ can be estimated directly by averaging all the q th design's replications, i.e. $\hat{\mu}_{d_q} = \frac{\sum r(d_q)}{\omega}$.

- $\hat{\sigma}_{d_q}^2$: the estimated variance of the design d_q . If $d_q \in \Omega^s$, $\hat{\sigma}_{d_q}^2$ can be estimated as $\frac{\sum (r(d_q) - \hat{\mu}_{d_q})^2}{\omega - 1}$.

The preceding definitions describe the variance and the mean as a way of evaluating the output performance measure of a design product (or a parameter's level) based on a statistical approach. As mentioned in the previous section, the output performance measure is used to calculate the probability of successfully achieving the requirement. The system range (the distribution of the output performance measure) is assumed to have a theoretical non-uniform probability distribution, whereas the design range (the functional requirement) is represented by a tolerance. Thus, the probability of satisfying the functional requirement is the area, under the theoretical probability distribution of the system range, which falls between the limits defined by the requisite tolerance.

In general, however, the theoretical distribution of the output performance measure is unknown. When this is the case, it is convenient to assume that the output performance measure r has the Normal distribution. Considering this assumption, next we formulate the success probability.

- P_{d_q} : the probability that a proposed design d_q successfully achieves the functional requirement, i.e. $P_{d_q} = P(r \in t / d_q)$. P_{d_q} is estimated as the probability that the design response of d_q falls within the predetermined tolerance, i.e. $P(r \in t / \hat{\mu}_{d_q}, \hat{\sigma}_{d_q}^2)$. Detailed expressions for computing P_{d_q} are presented in Appendix A.

To carry out the P-Learning algorithm, we need to define the following probabilities:

- $P_{\lambda_i^j}$: the probability that any design solution in which the j th level is assigned to parameter i successfully achieves the functional requirement, i.e.

$P_{\lambda_i^j} = P(r \in t / \lambda_i^j \in d)$. $P_{\lambda_i^j}$ is estimated as the probability that the response of

all designs in S_i^j falls within the specified tolerance, i.e. $P_{\lambda_i^j} =$

$P\left(r \in t / \hat{\mu}_{\lambda_i^j}, \hat{\sigma}_{\lambda_i^j}^2\right)$. $P_{\lambda_i^j}$ is termed as the *experimental success probability*. The

meaning of $P_{\lambda_i^j}$ is that it evaluates, based on the design matrix Ω^S , the probability

of successfully achieving the functional requirement for designs that have the j th level assigned to parameter i . Detailed expressions for computing $P_{\lambda_i^j}$ are presented

in Appendix A.

- $P(\lambda_i^j \in d)$: the probability that any design solution has the i th parameter assigned to its j th level. $P(\lambda_i^j \in d)$ is termed as the *preference probability*. As shown in the sequel, the P-learning algorithm generates the sample of candidate solutions, Ω^S , with a bias toward candidate designs as determined by the preference probabilities. Thus, $P(\lambda_i^j \in d)$ may be said to govern the percentage of designs in

Ω^S that have the level j assigned to their i th parameter (i.e. $P(\lambda_i^j \in d) \approx \frac{|S_i^j|}{|\Omega^S|}$), and

thus reflects the designer’s preference of which designs are to be included in Ω^S . In the “zero information” case ($S=1$), an orthogonal array for the *initial* design of experiments is constructed, and for each parameter and each level $P(\lambda_i^j \in d)$ is initially set to $\frac{1}{M}$. When more information is obtained (in subsequent steps), the preference probabilities are modified such that the designer assigns computational efforts to those designs that have higher probability of satisfying the functional requirements. The preference probabilities are modified in accordance with the *a posteriori preference probabilities* as described in the following.

- $P(\lambda_i^j \in d / r \in t)$, the *a posteriori preference probability*, is defined as the probability that the j th level is assigned to parameter i given that there exists a design that achieves the functional requirement. $P(\lambda_i^j \in d / r \in t)$ is computed by using Bayes’ theorem (see Appendix B), as follows:

$$P(\lambda_i^j \in d / r \in t) = \frac{\overbrace{P(\lambda_i^j \in d)}^{\text{Experimental}} \cdot \overbrace{P(\lambda_i^j \in d)}^{\text{Pr eferences}}}{\sum_j \{P_{\lambda_i^j} \cdot P(\lambda_i^j \in d)\}}$$

In the next step (S+1), the P-learning algorithm modifies the preference probabilities in accordance with the computed a posteriori preference probabilities (as described in the subsequent section).

13.3.3 THE P-LEARNING ALGORITHM

Algorithm P-Learning (Single performance measure)

1. $1 \leftarrow S$.
2. For each parameter $i \in I$ and each level $j \in J$, set $P(\lambda_i^j \in d) = 1/M$.
3. Initialization: set the initial representative sample Ω^1 to be the smallest orthogonal array with at least $\delta_1 \cdot M^N$ rows of experiments ($\delta_1 \leq 1$). δ_1 denotes the fraction of designs out of all the feasible designs.
4. Main Loop:
 - a) Simulate each design point in Ω^S for ω independent replications.
 - b) If $S=1$, identify the parameters that have the greatest effect on the performance measure by applying analysis of variance and pooling up techniques. If $S>1$, include all the parameters in subsequent steps.
 - c) For each main parameter $i \in I$ and each level $j \in J$, compute the estimated mean response $\hat{\mu}_{\lambda_i^j}$. If the sample size used to estimate $\hat{\sigma}_{\lambda_i^j}^2$ is small (e.g. ≤ 5), then set the variance $\hat{\sigma}_{\lambda_i^j}^2$ as the mean residual error (MSe) in the ANOVA table that was constructed in the first step ($S=1$). Otherwise compute $\hat{\sigma}_{\lambda_i^j}^2$ as shown earlier. Finally, compute $P_{\lambda_i^j}$.
 - d) For each parameter $i \in I$ and each level $j \in J$, compute the a posteriori preference probability $P(\lambda_i^j \in d / r \in t)$.
 - e) For each main parameter $i \in I$ and each level $j \in J$, update the preference probability by setting $P(\lambda_i^j \in d) = P(\lambda_i^j \in d / r \in t)$.
 - f) For each design point $d_q \in \Omega^S$, compute the success probability P_{d_q} (if the sample size used to estimate the variance $\hat{\sigma}_{d_q}^2$ is small, then set the

variance $\hat{\sigma}_{d_q}^2$ as the residual error in the ANOVA table. Otherwise compute

$\hat{\sigma}_{d_q}^2$ as shown earlier).

5. $S+1 \leftarrow S$.
6. Design matrix revision: generate $\delta_S \cdot |\Omega^{S-1}|$ designs ($\delta_S \leq 1$) such that for each design point and each parameter $i \in I$, the parameter i is fixed to its j th level with probability $P(\lambda_i^j \in d)$.
7. Termination criterion: IF C or more computer simulation runs have been conducted thus far THEN return the design solution that yields the maximum value of success probability among the design points generated thus far, and return the design solution that yields the maximum value; OTHERWISE, return to the Main Loop (STEP 4).

The P-learning algorithm is illustrated in Figure 13.2.

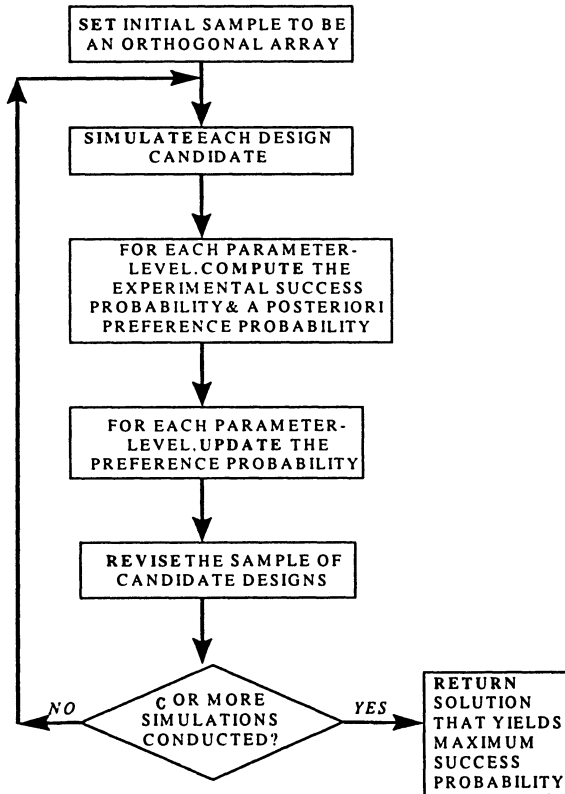


Figure 13.2 Flowchart of the P-Learning Algorithm

13.4 ILLUSTRATIVE EXAMPLE

Consider the following design problem where the parameter index set is $I = \{1, 2, 3, 4\}$; the level index set is $J = \{1, 2\}$; and the required functional requirement (tolerance) associated with the single output performance measure r is given by $T = \{r/r \geq 7.5\}$. The design space includes $2^4 = 16$ feasible design solutions. The sample reduction ratios are set to $\delta_1 = \delta_2 = 0.5$, and the termination criterion is set to $C = 12$.

- Initialization: Based on the reduction ratio δ_1 , when the designer has no information about which parameter-levels satisfy the governing requirements, the initial representative sample Ω^1 is set to be the L_8 orthogonal array presented in Figure 13.3. The L_8 orthogonal array guarantees a resolution level of 2, which means that all main parameters are unconfounded with two-parameter interactions, but some groups of two-parameter interactions are confounded (mixed) with each other. There are several ways to assign parameters to the columns of the L_8 orthogonal array. The assignment process, which is knowledge driven, is performed by a family of linear graphs [1]. The design points in Ω^1 were simulated, each for a single replication (i.e. $\omega = 1$). For each parameter $i \in I$ and each level $j \in J$, the preference probabilities $P(\lambda_i^j \in d)$ are set to $1/M = 0.5$.

test	λ_1	λ_2	$\lambda_1\lambda_2$	λ_3	$\lambda_1\lambda_3$	$\lambda_1\lambda_4$	λ_4	response
1	1	1	1	1	1	1	1	8
2	1	1	1	2	2	2	2	4
3	1	2	2	1	1	2	2	12
4	1	2	2	2	2	1	1	8
5	2	1	2	1	2	1	2	4
6	2	1	2	2	1	2	1	1
7	2	2	1	1	2	2	1	5
8	2	2	1	2	1	1	2	3

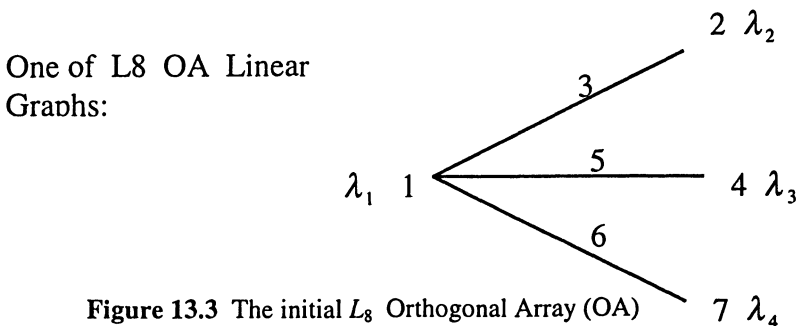


Figure 13.3 The initial L_8 Orthogonal Array (OA)

- The parameters (including two-parameter interactions) that have the greatest effect on the performance measure are identified by applying analysis of variance and pooling up techniques. Table 13.1 has rows for source of variation (main parameters and their interactions) and p-values for its source of variation produced by ANOVA. The large p-values, associated with $\lambda_1\lambda_2$ and λ_4 , indicate that there is not significantly real effect of the interaction $\lambda_1\lambda_2$ and the main parameter λ_4 . Thus, the residual error does not reflect all the unexplained sum of squares. Consequently, a different combination of column effects is required in order to better estimate the residual error. This type of column combination is determined by “pooling” strategies. Two pooling strategies can be considered for design [1]: “pooling down” versus “pooling up.”

Table 13.1 The ANOVA Table

*** ANALYSIS OF VARIANCE ***					
PERFORMANCE					
by $\lambda_1, \lambda_1\lambda_2, \lambda_2, \lambda_3,$ and λ_4					
UNIQUE sums of squares					
All effects entered simultaneously					
Source of Variation	Sum of Squares	DF	Mean Square	F	Sig. of F
Main Effects	84.625	5	16.925	27.080	.036
λ_1	45.125	1	45.125	72.020	.014
$\lambda_1\lambda_2$	3.125	1	3.125	5.000	.155
λ_2	15.125	1	15.125	24.20	.039
λ_3	21.125	1	21.125	33.80	.028
λ_4	.125	1	.125	.200	.698
Explained	85.625	5	16.925	27.080	.007
Residual	.250	2	.625		
Total	85.875	7	12.268		

8 cases were processed.
0 cases (.0 pct) were missing.

The *pooling down* strategy entails pooling all but the largest column effect, and F-testing the largest against the remaining pool together. If the column effect is significant, then the next largest is removed from the pool and F-testing is being performed again until some insignificant F ratio is obtained. The *pooling up* strategy

entails pooling the least column with the residual error and F-testing all other columns against the pooled columns. This process repeats itself until all the remaining columns are significant. The pooling up strategy tends to maximize the number of significant factors while the pooling down strategy tends to minimize them. In addition, the pooling up strategy tends to increase the alpha type of error, i.e. judging some factors to be significant while, in fact, they are not. On the other hand, the pooling down strategy increases the likelihood of a beta type of error, i.e. judging some significant factors to be non-significant while, in fact, they are. From the designer's perspective a beta type of error is less desirable, since once a factor has been judged to be insignificant, that factor is excluded from further steps of the P-learning algorithm. However, in case of an alpha type of error, the factor is included in further steps, which enables the designer to revise his judgments.

In the example, both strategies conclude with three significant parameters, $\lambda_1, \lambda_2, \lambda_3$ as shown in Table 13.2. However, in accordance with the P-learning algorithm, we include λ_4 in further steps.

Table 13.2 Final ANOVA Table After Conducting A Pooling Up Strategy

*** ANALYSIS OF VARIANCE ***					
PERFORMANCE by $\lambda_1, \lambda_2,$ and λ_3					
UNIQUE sums of squares All effects entered simultaneously					
Source of Variation	Sum of Squares	DF	Mean Square	F	Sig. of F
Main Effects	81.375	3	27.125	24.111	.005
λ_1	45.125	1	45.125	40.111	.003
λ_3	21.125	1	21.125	18.778	.012
λ_2	15.125	1	15.125	13.444	.021
Explained	81.375	3	27.125	24.111	.005
Residual	4.500	4	1.125		
Total	85.875	7	12.268		

8 cases were processed.
0 cases (.0 %) were missing.

Due to empty cells or a singular matrix,
higher order interactions have been suppressed.

- For each significant parameter $i \in I$ and each level $j \in J$, the estimated mean response $\hat{\mu}_{\lambda_i^j}$ is computed. For example, the estimated mean response $\hat{\mu}_{\lambda_1^1}$ and $\hat{\mu}_{\lambda_4^2}$ are computed as follows:

$$\hat{\mu}_{\lambda_1^1} = \frac{\sum_{q \in S_1^1} r^q}{|S_1^1| \cdot \omega} = \frac{r^1 + r^2 + r^3 + r^4}{4 \cdot 1} = \frac{8 + 4 + 12 + 8}{4} = \frac{32}{4} = 8, S_1^1 = \{d_1, d_2, d_3, d_4\}.$$

$$\hat{\mu}_{\lambda_4^2} = \frac{\sum_{q \in S_4^2} r^q}{|S_4^2| \cdot \omega} = \frac{r^2 + r^3 + r^5 + r^8}{4 \cdot 1} = \frac{4 + 12 + 4 + 3}{4} = \frac{23}{4} = 5.75,$$

$$S_4^2 = \{d_2, d_3, d_5, d_8\}.$$

Since the sample size used to estimate $\hat{\sigma}_{\lambda_i^j}^2$ is small, the variance $\hat{\sigma}_{\lambda_i^j}^2$ is set as the residual error in Table 13.2, i.e. $\hat{\sigma}_{\lambda_i^j}^2 = 1.125$. The estimated mean responses for all the significant parameters are summarized in Table 13.3.

Table 13.3 Mean Analysis

Summary of results by level of λ_1			
Variable	Value	Label	Cases
λ_1	1		4
λ_1	2		4
Summary of results by level of λ_2			
Variable	Value	Label	Cases
λ_2	1		4
λ_2	2		4
Summary of results by level of λ_3			
Variable	Value	Label	Cases
λ_3	1		4
λ_3	2		4
Summary of results by level of λ_4			
Variable	Value	Label	Cases
λ_4	1		4
λ_4	2		4
Variance in all cases is estimated as 1.125			
Total Cases = 8			

- Following the required functional requirement (tolerance) as given by $T = \{r/r \geq 7.5\}$, the experimental success probability $P_{\lambda_i^j}$ is computed for each significant parameter $i \in I$ and each level $j \in J$ (see Appendix A). For example, the experimental success probability $P_{\lambda_1^1}$ is computed as follows: Let $\nu_E=4$ denotes the total number of degrees of freedom used to estimate the residual error (MSe) as summarized in Table 13.2. Let $t_{\alpha,4} = \frac{(\hat{\mu}_{\lambda_1^1} - 7.5)}{\sqrt{MSe}} = \frac{(8 - 7.5)}{1.06} = 0.47$ is the critical point for the t distribution with 4 degrees of freedom, i.e. $P(t_4 \geq t_{\alpha,4}) = 1 - \alpha = 0.199$. Then $P_{\lambda_1^1} = 1 - \alpha = 0.8004$. In a similar manner one obtains the following experimental success probabilities: $P_{\lambda_1^2} \approx 6.57E-4$; $P_{\lambda_2^1} \approx 1.79E-3$; $P_{\lambda_2^2} \approx 0.1995$; $P_{\lambda_3^1} \approx 0.33095$; $P_{\lambda_3^2} \approx 1.36E-3$; $P_{\lambda_4^1} \approx 1.958E-2$; $P_{\lambda_4^2} \approx 0.0299$.

- For each significant parameter $i \in I$ and each level $j \in J$, the preference probability is updated by setting $P(\lambda_i^j \in d) = P(\lambda_i^j / r \in t)$. Thus, the revised preference probability $P(\lambda_1^1 \in d)$ is computed as follows:

$$P(\lambda_1^1 / r \in t) = \frac{\overbrace{P_{\lambda_1^1}^{Experimental}} \cdot \overbrace{P(\lambda_1^1 \in d)^{Preferences}}}{\sum_j \{P_{\lambda_1^j} \cdot P(\lambda_1^j \in d)\}} = \frac{0.8004 \cdot 0.5}{0.8004 \cdot 0.5 + 6.57 \cdot 10^{-4} \cdot 0.5} \approx 1. \quad \text{In a}$$

similar manner one obtains the following revised preference probabilities: $P(\lambda_1^2 \in d) = 0$; $P(\lambda_2^1 \in d) = 0.01$; $P(\lambda_2^2 \in d) = 0.99$; $P(\lambda_3^1 \in d) = 0.996$; $P(\lambda_3^2 \in d) = 0.004$; $P(\lambda_4^1 \in d) = 0.39$; $P(\lambda_4^2 \in d) = 0.61$.

- In the next step ($S=2$), the designer generates $\delta^2 \cdot |\Omega^1| = 0.5 \cdot 8 = 4$ new designs such that the parameter i is assigned to its j th level with the revised preference probability $P(\lambda_i^j \in d)$. The resulting representative sample Ω^2 is described in Table 13.4. For each design point $d_q \in \Omega^2$ its success probability P_{d_q} is computed, and the design point $d_q = (1,1,1,2)$ that yields the maximum value is selected as the design solution.

Table 13.4 The Design Matrix Ω^2

test	λ_1	λ_2	λ_3	λ_4	response
1	1	2	1	2	12
2	1	2	1	1	10
3	1	1	1	2	15
4	1	1	1	1	8

13.5 A CATALOGUE STRUCTURE FOR THE P-LEARNING ALGORITHM

The adaptive learning methodology can be supported by capturing the context and history of previous design processes. Information and data that are obtained at the termination of the P-learning algorithm can be collected based on the field of interest and stored in a catalogue for future redesign activities. The data used in the catalogue include (in each row): a list of parameter-level combinations (i.e. λ_i^j), the sample size (n_{eff}) used to estimate the mean response of all designs in S_i^j , the estimated mean ($\hat{\mu}_{\lambda_i^j}$) response of all designs in S_i^j , the estimated variance ($\hat{\sigma}_{\lambda_i^j}^2$) of all designs in S_i^j , the preference probability $P(\lambda_i^j \in d)$, the experimental success probability $P_{\lambda_i^j}$, the a posteriori preference probability $P(\lambda_i^j \in d / r \in t)$, and the desired set of specifications t . Table 13.5 presents the catalogue associated with the example provided in Section 13.4.

Design catalogues support incremental redesign activities by capturing the context and history of a particular design process in order that the experience (good or bad) may be reused in future redesign activities. For example, (1) if the perceived needs associated with the output performance measures are changed (shifting to a new t), the last two columns in the design catalogue (using the Table 13.5 format) would be modified to reflect the changes, and the P-learning algorithm would be initiated with a new sample of candidate solutions generated according to the revised preference probabilities in the last column; (2) any revision of the preference probabilities due to newly acquired information would be reflected in the fourth and last columns; and (3) if the perceived needs are modified to include conjunctions over previously defined (and independent) functional requirements (e.g. “work in process should be between 6 to 10 units” and “number of defective units should be less than 100 parts per million”), then the designer can revise the preference probabilities by simply multiplying the “experimental success probabilities” columns ($P_{\lambda_i^j}$) in the corresponding catalogues.

Table 13.5 A Catalogue Structure for A Required Functional Requirement (Tolerance) Associated with the Single Output Performance Measure r as given by $t = \{r/r \geq 75\}$.

λ_i^j	n_{eff}	$\hat{\mu}_{\lambda_i^j}$	$P(\lambda_i^j \in d)$	$\hat{\sigma}_{\lambda_i^j}^2$	$P_{\lambda_i^j}$	$P(\lambda_i^j \in d / r \in t)$
λ_1^1	4	8.00	0.5	1.125	0.8004	99.92%
λ_1^2	4	3.25	0.5	1.125	6.57E-4	0.08%
λ_2^1	4	4.25	0.5	1.125	1.79E-3	0.89%
λ_2^2	4	7.00	0.5	1.125	0.1995	99.11%
λ_3^1	4	7.25	0.5	1.125	0.33095	99.59%
λ_3^2	4	4.00	0.5	1.125	1.36E-3	0.41%
λ_4^1	4	5.50	0.5	1.125	2.37E-2	39.55%
λ_4^2	4	5.75	0.5	1.125	6.62E-3	60.45%

13.6 SUMMARY

In Chapter 8, a functionality complexity measure was provided as a rational means for quantifying how well a proposed design satisfies the governing requirements. In this chapter, the functionality complexity measure is shown to also have a heuristic value by presenting a method for adaptive learning of successful designs. The P-learning algorithm constructs a sequence of samples in which each sample includes particular designs to simulate. Given such a sample, the P-learning algorithm learns which factors appear to satisfy the governing requirement in terms of the overall success probability. As the desired information is obtained, the P-learning algorithm generates new candidate solution with a bias toward candidate solutions that include better levels. In Chapter 7, we linked between the probabilistic search strategy as presented in this chapter and Information Theory.

The P-learning algorithm can be used when there is more than one requirement to be satisfied. This is done by introducing a heuristic measure which quantifies how well a proposed design satisfies the overall probability of successfully achieving a set of independent design requirements. The P-learning algorithm can be modified in order to account for dependent requirements by estimating *conditional* success probabilities. However, considering the simplicity of the P-learning algorithm and its performance, we believe that the proposed functionality complexity measure is suitable.

In Chapter 19, we provide a detailed case study of a flexible manufacturing system design by applying the P-learning algorithm.

APPENDIX A - COMPUTATION OF THE EXPERIMENTAL SUCCESS PROBABILITIES

In this appendix we show how to compute the experimental success probability $P_{\lambda_i^j}$. To compute the design success probability P_{d_q} , we simply use the respective statistics when needed (i.e. $\hat{\mu}_{d_q}$ and $\hat{\sigma}_{d_q}^2$). Three cases are considered in accordance with the form of the functional requirement.

Case 1 - $T = \{r / r \geq LB\}$ or $T = \{r / r \leq UB\}$:

Let ν_E denotes the of number of degrees of freedom used to estimate the residual error (MSe). Let $t_{\alpha, \nu_E} = \frac{(\hat{\mu}_{\lambda_i^j} - LB)}{\sqrt{MSe}}$ be the critical point for the t distribution with ν_E degrees of freedom such that $P(t_{\nu_E} \geq t_{\alpha, \nu_E}) = 1 - \alpha$. Then $P_{\lambda_i^j} = 1 - \alpha$ as described in Figure 13.3.

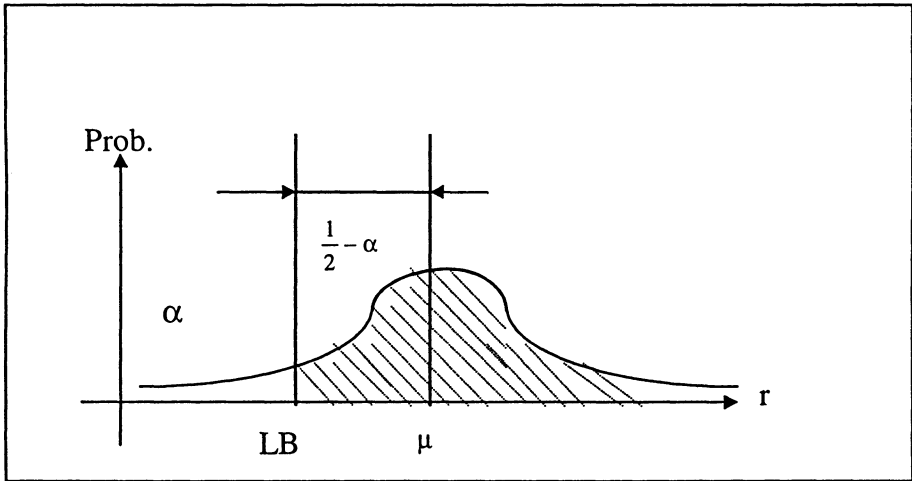


Figure 13.3 Experimental Success Probability - Case 1

Case 2 - $T = \{r / r \leq UB\}$:

To compute the experimental success probability in this case, we let $t_{\alpha, \nu_E} = \frac{(UB - \hat{\mu}_{\lambda_i^j})}{\sqrt{MSe}}$ be the critical point for the t distribution with ν_E degrees of freedom

such that $P(t_{vE} \leq t_{\alpha, vE}) = 1 - \alpha$. Then $P_{\lambda_i^j} = 1 - \alpha$.

Case 3 - $T = \{r / LB \leq r \leq UB\}$:

Let $t_{\alpha_{LB}, vE} = \frac{(\hat{\mu}_{\lambda_i^j} - LB)}{\sqrt{MSe}}$ and $t_{\alpha_{UB}, vE} = \frac{(UB - \hat{\mu}_{\lambda_i^j})}{\sqrt{MSe}}$ be defined as in Case 1 and Case 2. Then $P_{\lambda_i^j} = 1 - \alpha_{UB} - \alpha_{LB}$, as described in Figure 13.4.

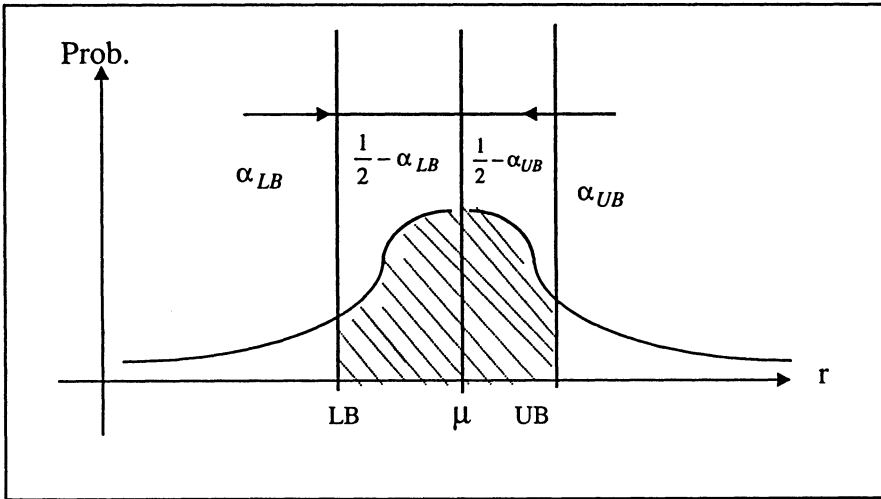


Figure 13.4 Experimental Success Probability - Case 2

APPENDIX B - BAYES' THEOREM

The P-algorithm uses Bayes' Theorem to update the preference probabilities at each step. Despite its simplicity, Bayes' Theorem is widely used in biometrics, epidemiology, and communications theory.

Bayes' Theorem: Let $A_i, i = 1, \dots, n$ be a set of disjoint and exhaustive events defined on \mathcal{E} . Then $\bigcup_{i=1}^n A_i = \Omega, A_i \cap A_j = \phi \ i \neq j$. Let B be any event defined on \mathcal{E} , with $P(B) > 0$. With $P(A_i) \neq 0$ for all i :

$$P(A_j / B) = \frac{P(B / A_j)P(A_j)}{\sum_{i=1}^n P(B / A_i)P(A_i)} .$$

In our case, we define the following events:

$B \equiv \{ \text{the output response falls within the tolerance, i.e. } r \in t \}$

$A_i \equiv \{ \text{parameter } i \text{ is set to its } j\text{th level, i.e. } \lambda_i^j \in d \}$

Then

$$P(B / A_j) = P (r \in t / \lambda_i^j \in d) = P_{\lambda_i^j} \quad (\text{the experimental success probability})$$

$P(A_j / B) = P (\lambda_i^j \in d / r \in t) = P (\lambda_i^j / r \in t)$ (the a posteriori preference probability).

REFERENCES

1. Ross, P. J., *Taguchi Techniques for Quality Engineering*, McGraw-Hill, 1988.

CHAPTER 14

MAINTAINING CONSISTENCY IN THE DESIGN PROCESS

In Chapter 5, we investigated the notion of design consistency: small changes in specifications should lead to small changes in design. The mathematical concept that is used to investigate the principle of design consistency is that of continuous analysis and continuous synthesis. In this chapter, the concept of design consistency in the area of variational design is further formalized, and the COAST (COnsistency through Analysis of Solution Trajectories) methodology is implemented for maintaining design consistency in those design areas where similarity between designs can be calculated [21]. The formal description of an evolutionary design model as given in Chapter 6 is used to define the design paradigm used in COAST.

In variational design, the dimensions of a part are calculated by solving a system of constraints (typically, nonlinear equations). Two characteristics of variational design combine to necessitate the development of rigorous design consistency techniques: (1) systems of nonlinear equations often have multiple solutions, and (2) design is an incremental, evolutionary process. If design consistency is not maintained, the designer can be confronted with manually correcting the design at each iteration of the design process.

When systems of constraints are modified, the COAST method maintains design consistency by following the homotopy of the desired solution between the original and modified systems of constraints. In this chapter, a design consistency is defined numerically and interval-based continuation methods (such as that used in the COAST method) are analyzed and proven to either guarantee design consistency or detect when it cannot do so. Finally, to demonstrate the COAST methodology, a cantilever beam is designed and representative solution trajectories are given for each step in the design process.

14.1 INTRODUCTION

14.1.1 VARIATIONAL DESIGN

In computer-aided design systems, a design is represented in terms of its geometry.

In constraint-based design, the designer is able to describe the geometry of the part using relations (e.g. distance and angle). This is in contrast to systems in which the user must explicitly define the geometry of the design.

Constraint-based design systems are divided broadly into two categories: parametric and variational [14]. In parametric design systems, the designer explicitly defines the value of each dimension of a part as it is created. The value may take the form of a numerical constant or a mathematical expression that reference attributes of previously defined parts, but it must be completely defined. The design system records each step of the design and creates a design sequence. When modifications are needed, they are made to a specific step in the design sequence and the sequence is “replayed”, updating every dependent design step. While this is a powerful paradigm for constraint-based design, it suffers from requiring the user to explicitly constrain the value of each dimension. This limits the type of constraint that can be implemented.

In variational design, the entire system of constraints for a part or parts is solved simultaneously with a constraint solver (e.g., Newton-Raphson). Variational design systems offer several advantages including the ability to solve problems whose dimensions cannot be calculated with a single algebraic expression as well as subsuming most of the capabilities of parametric design systems. An entire design sequence in a parametric design system can be thought of as a system of constraints in a variational design system.

Despite its advantages, the numerical approach to constraint management has been often criticized for the multitude of solutions that are possible and the lack of a good method for distinguishing among them besides requiring the designer to input an initial guess for the design whenever they enter or modify constraints.

14.1.2 DESIGN CONSISTENCY IN VARIATIONAL DESIGN

The cost of making changes to a design depends heavily on the ability of the design software to maintain consistency. Specifically, we consider the problem of maintaining consistency in variational design systems where a part is designed by creating a system of constraints that describe the dimensions of the part. Furthermore, we consider a numerical approach to constraint satisfaction in which the constraints are transformed into a system of algebraic equations (typically nonlinear) and solved simultaneously. Systems of equations have historically been solved using iterative numerical techniques such as the Newton-Raphson method. One common critique of such numerical constraint satisfaction techniques is that when the equations are nonlinear, there are often multiple competing solutions to the system, but only one is usually acceptable to the designer. Simple iterative techniques for solving the system of constraints then fall short due to their lack of control over which solution the system converges to. Because of the iterative nature of design, modifications will certainly occur whereupon the lack of control translates into much added effort for the designer in repeatedly guiding the constraint solver to the correct solution.

The problem of converging to an unintended solution arises because anytime a system of constraints is modified, it is typically completely re-solved as if it were a new problem. Rather than treating each new system of constraints as completely new, we define a convex homotopy from the original system of constraints to the new system of constraints. An interval-based continuation method, COAST (CONSistency through Analysis of Solution Trajectories) is used to deterministically track the solution trajectory of the desired solution as it is modified through the homotopy.

14.1.3 CHAPTER OUTLINE

The remainder of this chapter is organized as follows. In Section 14.2, we introduce previous research in design consistency, from both geometric and numerical approaches. In Section 14.3, we provide a general definition of design consistency. In Section 14.4, we define our variational design paradigm. In Section 14.5, we derive mathematical definitions and theorems about design consistency and prove the design consistency capability of COAST. In Section 14.6, we introduce a simple interval-based continuation method algorithm (COAST) for following a desired solution curve. In Section 14.7, to demonstrate the COAST methodology, a cantilever beam is designed and representative solution trajectories are given for each step in the design process. In Section 14.8, we conclude and describe future work to the COAST algorithm.

14.2 PREVIOUS EFFORTS

In variational design, when the designer modifies the system of constraints, there are two primary methods for ensuring that the new design is consistent with the previous design: either the program can use logic to derive the correct solution from the geometry of the design or the program can use mathematical techniques to converge to the correct solution.

14.2.1 GEOMETRIC REASONING

Geometric techniques seek to exploit geometric information (either defined by the designer or inherent in the design) to characterize the possible solutions and maintain a consistent solution.

In geometrical CAD systems, where the constraints are all geometric relations, it is often possible to characterize solutions by observing certain topological properties of the objects. For instance, [20] introduces two relative position constraints, *order_on* and *side_of*. The first one allows the user to define the order of certain objects as they lay on other objects. For instance, if two points are constrained to lay on a line and be a certain distance from one another, there is an ambiguity as to their order. The second constraint allows the user to define on which side an object lies.

For instance, if a line is constrained to be parallel to another line, there is an ambiguity as to which side of the line to draw the parallel line. Both these types of ambiguities are handled by those additional constraints. Similarly, when [3] allow constraints which describe the slope of points on a curve, there is ambiguity introduced (e.g., there are typically two lines that are tangent to a circle with a given slope). This ambiguity is removed by characterizing the two slopes according to the \pm solutions of the quadratic formula used to calculate the slopes.

Rather than characterizing the different possible solutions or adding relative constraints, there are researchers such as [10] who define design consistency to mean that the topology and the geometry of a part are in agreement. Consequently, they describe techniques to calculate the topological and geometrical genus of the object and if they are the same, the design is consistent. They cannot, however, discern between multiple feasible solutions.

14.2.2 NUMERICAL TECHNIQUES BASED ON CONTINUATION METHODS

Consider a system of m equations that describe the values of n unknown variables. The system is said to be *well-constrained* if $m = n$ and there are a finite number of solutions:

$$\underline{f}(\underline{d}) = \underline{b} \Rightarrow \underline{d} = \{ \underline{d}_1, \underline{d}_2, \dots, \underline{d}_i^*, \dots, \underline{d}_{p_1} \}.$$

When the system of equations is first defined, there is no general approach for deciding which of the solutions is the correct one - this is based on the designer's intent. In order to identify the correct solution, variational design systems typically rely on the user to enter an initial guess, which is used as the initial solution for an iterative constraint solver. Then, when the designer decides to change the system of constraints by either choosing new constraints and/or defining new constraint values, not only does the desired solution change, but the entire set of possible solutions changes:

$$\underline{f}(\underline{d}') = \underline{b}' \Rightarrow \underline{d}' = \{ \underline{d}'_1, \underline{d}'_2, \dots, \underline{d}'_j^*, \dots, \underline{d}'_{p_2} \}.$$

Continuation methods are techniques for following a curve. In this case, the curve is the path that the solution follows as $(\underline{b}' - \underline{b})$ is increased. If $(\underline{b}' - \underline{b})$ is small, then \underline{d}' should be fairly close to \underline{d} . As $(\underline{b}' - \underline{b})$ becomes larger, \underline{d}' moves farther away from \underline{d} along the solution curve. If $(\underline{b}' - \underline{b})$ is chosen small enough, then \underline{d} can be an accurate initial guess into an iterative numeric constraint solver to solve for \underline{d}' .

Fixed Step Continuation

The basic continuation method consists of dividing up the changes to the constraint

values, $(\underline{b}' - \underline{b})$ into smaller steps. When the user changes the values of any of the constraints, $\underline{f}(\underline{d}) = \underline{b}'$, they can also enter the number of iteration steps to take, m . The technique then iteratively steps towards the new value solving the system of equations, $\underline{f}(\underline{d}) = \underline{b} + \frac{(\underline{b}' - \underline{b})}{m} \cdot \beta$, $\beta = 1, \dots, m$, at each increment of the constraint values, using the converged value at each iteration step, β , as the initial solution of the next iteration [12, 13]. This is one of the first known uses of continuation methods for maintaining design consistency in a variational design system. It is not automatic, however, as the user must manually select a proper value of m in that is large enough to converge to the correct solution but not so large that it overly slows down the computation time.

Solution Characterization

Once all the solutions have been calculated for a given system of equations, continuation methods can be used to track all the solutions as they move to their new values. This technique is especially well developed for polynomial systems of moderate size and can greatly speed up the computation time over finding all the solutions without having a starting system. Once all the solutions are found, however, it still remains necessary to select the correct one. This is only possible if all the solutions can be characterized. For example, in [18], the authors use continuation methods to determine all the possible solutions to problems such as the inverse kinematics of a 6-revolute-joint manipulator as the joint angles are varied. This technique, however, requires the ability to characterize all the possible solutions and it is really only suited for systems of polynomial equations.

Advanced Continuation

In order to improve the robustness of continuation techniques, several more advanced techniques are outlined in [2]. These include: (1) using arc length as the parameter of the curve, (2) incorporating higher-order gradient information to create a more accurate prediction, and (3) using different corrector methods (besides Newton-Raphson). Because all these techniques are based on the predictor-corrector model, however, it is still possible to converge to a wrong solution and there would be no way to determine when this happens (besides requiring the user to recognize it), as demonstrated in [9].

14.2.3 OTHER NUMERICAL TECHNIQUES

Prior research efforts in variational design have all had to be able to re-converge to a consistent solution when a system of constraints is modified. Six different methods

identified in the literature are summarized below.

User-Defined Initial Solution

Methods for solving a system of nonlinear equations, $f(\underline{d}) = \underline{b}$, are all of the iterative form $\underline{d}_{k+1} = \underline{d}_k + F(\underline{d}_k)$. In the Newton-Raphson method, $F(\underline{d}_k) = -J(\underline{d}_k)^{-1}$. Every method for solving systems of nonlinear equations depends upon an initial solution, \underline{d}_0 , to control the specific solution it converges to. As such, every variational design program offers the user the ability to modify it [1, 3, 12, 17]. This is the most user-intensive method as it not only requires the user to constantly monitor the convergence and recognize when a spurious solution has been found, but also to know roughly where the new solution should be.

Relaxation Factor

Rather than requiring the user to specify the exact number of iteration steps to take for each parameter change, this method allows the user to simply slow down the convergence by multiplying the corrective vector by a scalar number (the relaxation factor): $\underline{d}_{k+1} = \underline{d}_k + \lambda F(\underline{d}_k)$ where λ typically varies between 0 and 1 [12, 13, 6]. While the user still has to monitor the convergence and recognize a spurious solution, it is similar to the fixed-step continuation method in that the user only has to decrease λ until the correct solution is found. As in the fixed-step continuation method, the user must tradeoff the probability of finding the correct solution the first time with converging quickly.

Curve-Crawl Factor

Instead of multiplying the correction vector by a scalar number, this method uses a function of the magnitude of the correction vector of the previous iteration, $\underline{d}_{k+1} = \underline{d}_k + C(F(|\underline{d}_{k-1}|))F(\underline{d}_k)$. Ideally, this function would become close to 1 as the magnitude of the previous correction vector headed towards 0 (thereby not limiting a slow convergence) and would become smaller for larger magnitudes of the previous correction vector in order to slow down a faster convergence [12, 13]. The same limitations as the relaxation factor method apply here.

Feasibility Limits

Rather than relying solely on the user to recognize when the constraint solver has

converged to an unintended solution, an alternative method is to assign feasibility limits to every variable [6]. If the constraint solver converges to a value outside the feasible range, the relaxation factor is decreased (slowing the convergence down) and the convergence is repeated until either the variables remain inside their feasibility limits or a given number of attempts has been exhausted. In the case of multiple feasible solutions, the user must again recognize the situation and adjust the convergence.

Manual Selection

This method finds every solution to the system of equations and then allows the user to select the desired solution. This approach is used in higher-level symbolic mathematical packages (e.g., Mathematica) and is stated as a future goal of [5]. As all solutions are considered equal in worth, there is no attempt to distinguish a correct solution from an unintended one. Besides the computational cost of finding all solutions, it requires user interaction after every constraint modification.

Over-Constraining the System

The final method is to have the user over-constrain the system until it is uniquely-constrained [5]. As this may sometimes yield no solutions, [4] and [19] additionally give the user the option to prioritize the constraints (rather than requiring them all to be simultaneously satisfied). Essentially, this technique allows the user to characterize the solutions. Characterization of solutions to a system of nonlinear equations is not possible in general, however. Additionally, if the designer doesn't characterize the solutions properly (allowing for every possible design scenario), they can lose the desired solution or even converge to a wrong solution.

14.2.4 DISCUSSION

In order to maintain a consistent design, the techniques discussed here either require the user to interact with the solver, guiding its convergence towards the correct solution, or to input enough extra information that there is only one solution and, hence, no ambiguity. None of the techniques are satisfactory for our system, however. When any of the constraints are modified, the user should not have to monitor the convergence and modify it when it converges to an unintended solution. Also, it is sufficiently challenging for the user to fully-define any system of constraints that requiring the user to over-constrain the system so that there is only one solution for all possible constraint values is not practical in general.

14.3 DESIGN CONSISTENCY

Because of the highly iterative nature of diagonalized design, maintaining design consistency is very important - so much so that design consistency is simply assumed by any designer. When the designer makes an incremental change to the design, they expect that the resulting design will be consistent with the beginning design. It is unreasonable to expect the designer to enter so many constraints that there is exactly one solution to the system of constraints. The initial solution was chosen because it met every constraint of the designer (both explicit and assumed). After a design modification, the constraint solver should honor the initial design choice. Additionally, as the design evolves, different groups (e.g., marketing, manufacturing, and accounting) each input different requirements for the final design. Since consensus is difficult to achieve, however, each design modification needs to be done consistently. Maintaining design consistency is easy for simple constraints in which every design parameter is explicitly defined with no ambiguity. However, as more powerful, implicit constraints are used, the problem of multiple solutions is introduced.

Morphology is the study of how objects change throughout their life. For example, in the area of zoology, spirals occurring in nature (e.g., mollusk shells or ram horns) can be categorized and their progression throughout the life of the creature can be described mathematically [7]. Similarly, it is asserted that in the typical design process, a design moves from one solution to another as it progresses throughout the life of the design cycle. How a given design moves between solutions can be understood and this information used to maintain the same, consistent solution.

More formally, at each design iteration, we begin with a current list of specifications, θ_0 , a set of possible solutions, $\{M\}_0$, and a specific satisfactory design, $M_0 \in \{M\}_0$. The specifications are then modified, $\theta_1 = \theta_0 + \Delta\theta_0$, leading to a new set of possible satisfactory designs, $\{M\}_1$:

$$\begin{array}{ccc} \theta_0 & \Rightarrow & M_0 \in \{M\}_0 \\ \downarrow & & \\ \theta_1 = \theta_0 + \Delta\theta_0 & \Rightarrow & \{M\}_1 \end{array}$$

At each iteration, we begin from a satisfactory design. When the specifications are modified, we wish to not only find a new satisfactory design, we wish to find the *intended* design. This is what is meant by *consistent design*. The problem becomes selecting the correct solution, $M_1 \in \{M\}_1$. If there is only one possible solution to the specifications, then it is easy to maintain a consistent design. It is much harder if there are multiple competing solutions, all of which satisfy the specifications. Fortunately, this iterative view of design directs us towards a principle of design consistency: *small changes in specifications should lead to small changes in design*.

Furthermore, large changes in specifications can often be decomposed to a series of small changes, in which case the principle can still be applied. With this principle, we define consistency between M_1 and M_0 recursively as follows:

Definition 14.1: M_1 (satisfying $\theta_1 = \theta_0 + \Delta\theta_0$) is *consistent* to M_0 (satisfying θ_0) if either:

- (1) M_1 is consistent to M^* (satisfying θ^*) and M^* is consistent to M_0 (transitivity condition), or
- (2) $\Delta\theta_0 = \epsilon_\theta$,

M_1 minimizes $\| \{M\}_1, M_0 \|$, and

$$\|M_1, M_0\| < K$$

where:

- $\|M_1, M_0\|$ is a metric that quantifies the “difference” between two designs (smaller values imply increasing similarity). This metric need only be meaningful for designs of sufficiently small difference (as opposed for any two general designs).
- ϵ_θ is any small change in specifications towards θ_1 such that there exists a symmetrical neighborhood about M_0 in which it is known that there is exactly one design solution for any change in the specifications, ϵ_θ^* ($0 < \epsilon_\theta^* \leq \epsilon_\theta$). Note that this also requires a metric to determine the “size” of a specification change and “similarity” between sets of specifications.
- K is a constant. Design should not be a chaotic environment (where small changes in input lead to large changes in output). As such, at very small increments of the specifications, there is some limit, K , to the acceptable size in the change in design, $\|M_1, M_0\|$.

From this definition, we can describe an algorithm to find the solution, M_1 , that is consistent to M_0 (see Figure 14.1). The individual algorithm steps shown in the figure are described more fully as follows:

Step 1: $\theta_{\delta_1} = \theta_0$, $M_{\delta_1} = M_0$, $\theta_{\delta_2} = \theta_0 + \epsilon_\theta$ (where ϵ_θ is defined as above)

Step 2: Calculate the solution, M_{δ_2} , that minimizes $\| \{M\}_{\delta_2}, M_{\delta_1} \|$

Step 3: $M_{\delta_1} = M_{\delta_2}$, $\theta_{\delta_1} = \theta_{\delta_2}$, $\theta_{\delta_2} = \theta_{\delta_2} + \epsilon_\theta$ (where ϵ_θ is defined as above)

Step 4: M_{δ_2} is the new, consistent solution

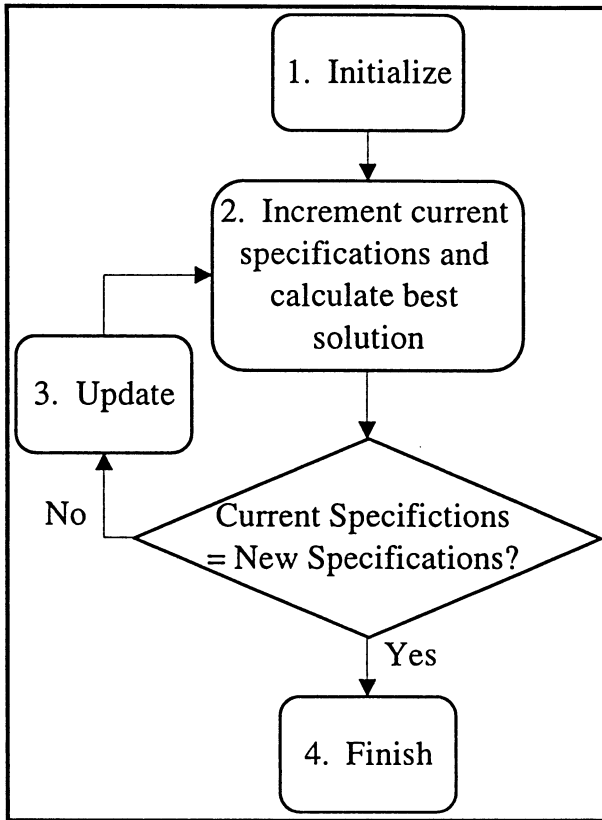


Figure 14.1 General Consistency Algorithm

14.4 DESIGN EVOLUTION IN VARIATIONAL DESIGN SYSTEMS

A *design* is defined in terms of parts and relations. Every part is fully described by a set of *dimensions* and can be uniquely identified by defining all values of its dimensions. A *configuration* is a complete set of dimension values, \underline{d} (fully describing a feasible part). An *attribute* is an algebraic expression in terms of the dimensions that describes a certain aspect of the part. A *constraint* consists of either a dimension or an attribute and a value that it must attain. Finally, a *requirement* consists of a qualitative description of the part which must be achieved for a satisfactory design.

Since variational design systems allow the user to enter all the required constraints before solving the system, it may appear that design becomes a single event rather than an *evolutionary* process. We argue that this is not the case. The designer cannot possibly consider all the possible constraints at once; they may want

to test a constraining scheme before the whole design has been constrained or new information or requirements may be introduced or modified after a design was supposedly completed. In this section, we formally describe the design paradigm as evidenced in the COAST methodology.

In Chapter 6, we provided a formal description of an evolutionary design model. A design *execution* is a sequence of *synthesis states*, each state being characterized by a design pair, $\langle \theta, M \rangle$, where:

- θ is the current design specification (here, the system of constraints and the set of requirements, $\theta = \theta^C \cup \theta^Q$) and
- M is the current design configuration (here, the dimension values which are calculated by solving θ^C).

The design process can thus be represented as a sequence of synthesis states, $\{\langle \theta_0, M_0 \rangle, \langle \theta_1, M_1 \rangle, \dots, \langle \theta_k, M_k \rangle\}$ where $\langle \theta_k, M_k \rangle$ is the termination state. Each transition between synthesis states is described by either termination (success or failure depending on whether or not M_k satisfies θ_k^Q) or modification of either the system of constraints, θ^C , or the set of requirements, θ^Q .

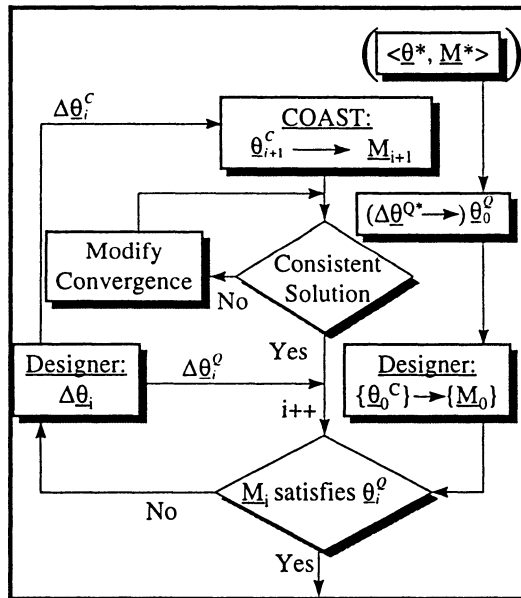


Figure 14.2 Design Loop

The evolutionary design process as viewed by the COAST methodology is summarized in the flowchart in Figure 14.2. The specifications, θ_i , are composed of

a set of requirements, $\underline{\theta}_i^Q$, and a system of constraints, $\underline{\theta}_i^C$. In all cases, $\underline{\theta}_i^C$ must contain as many equations as there are dimensions ($|\underline{\theta}_i^C| = |\underline{M}_i|$). The design process optionally begins with a successful design which meets some historical requirements, $\langle \underline{\theta}^*, \underline{M}^* \rangle$, and a change in the requirements, $\Delta \underline{\theta}^{Q*}$ that motivates the beginning of the design process. Alternatively, the design can start from an original set of requirements, $\underline{\theta}_0^Q$. The designer then creates a system of constraints, $\langle \underline{\theta}_0^C, \underline{M}_0 \rangle$, that will hopefully create a design that meets all the requirements. If \underline{M}_0 satisfies $\underline{\theta}_0^Q$, then the design is finished. Otherwise, the user iteratively modifies the constraints, $\Delta \underline{\theta}_{i+1}^C$, and/or the requirements, $\Delta \underline{\theta}_{i+1}^Q$, until a satisfactory design is found. It is the designer's job to modify the constraints and requirements however they feel is necessary to find a satisfactory design; it is the purpose of COAST to maintain a consistent solution to every design modification the designer performs.

Example 14.1: In order to demonstrate the nomenclature and solution trajectories, let us consider the very simple case of creating a point at a certain distance from two other points. The unknown point has two dimensions: the x and y coordinates. Additionally, let there be two attributes: distance from the point at (0,0) and distance from the point at (3,4). The two attributes are each equations in terms of the dimensions: $\sqrt{x^2 + y^2}$ and $\sqrt{(x-3)^2 + (y-4)^2}$, respectively. The user begins by specifying an initial guess of the desired point:

$$\begin{array}{l} \underline{\theta}_0^C: \quad \underline{M}_0: \\ \left\{ \begin{array}{l} x = 1 \\ y = 4 \end{array} \right. \rightarrow \left\{ \begin{array}{l} x = 1 \\ y = 4 \end{array} \right. \end{array}$$

The values of the attributes are calculated and the point is found to be distances 4.12 and 2 from (0,0) and (3,4), respectively. The user then decides that the point should be distances 3 and 6 from (0,0) and (3,4). The two attributes are solved simultaneously yielding the solution, (-2.51, 1.64):

$$\begin{array}{l} \underline{\theta}_1^C: \quad \underline{M}_1: \\ \left\{ \begin{array}{l} \text{distfrom}(0,0) = 3 \\ \text{distfrom}(3,4) = 6 \end{array} \right. \rightarrow \left\{ \begin{array}{l} x = -2.51 \\ y = 1.64 \end{array} \right. \end{array}$$

The two distance attributes at the beginning of the problem can be interpreted as in Figure 14.3-A. When the constraints are modified, however, there are two possible solution (see Figure 14.3-B).

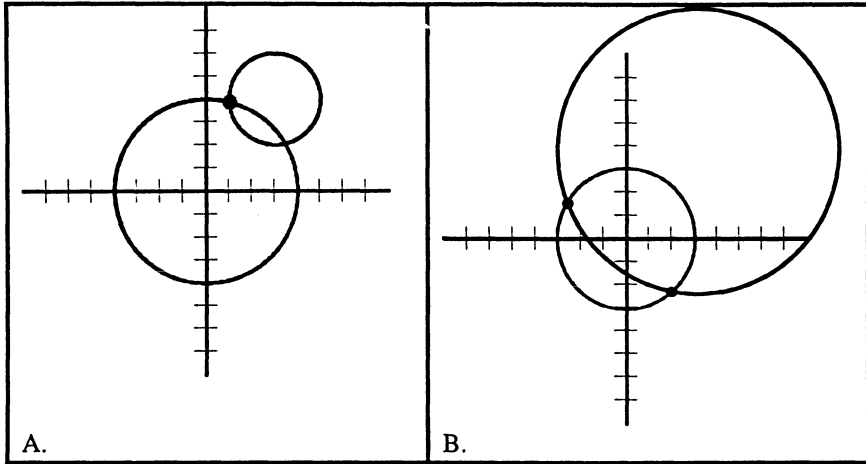


Figure 14.3 Simple Variational Design Example

If the original and new sets of parameter values are considered as two discrete events, the only way to maintain consistency of the design is to be able to distinguish individual solutions and be able to recognize which solution the user desires whenever the parameters are changed. While this is certainly possible for a simple geometric design problem, it is not possible in general for a system of nonlinear equations.

Rather than treating parameter changes as discrete events, we consider the continuous trajectory the desired solution follows as the constraints are changed from the original system to the new system. Rather than trying to characterize the solutions, the COAST method follows a specific solution as it moves along its solution trajectory. For instance, as the distance radii of the circles are changed, the desired point creates a trajectory from its original position to its new position (as shown in Figure 14.4).

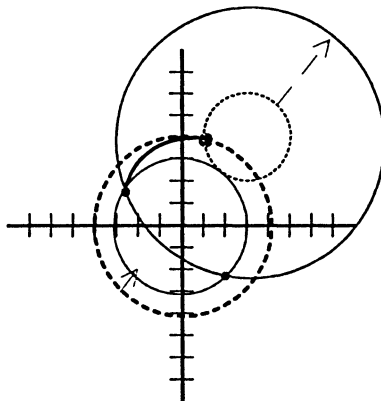


Figure 14.4 Solution Trajectory

14.5 DESIGN CONSISTENCY THROUGH SOLUTION TRAJECTORIES

Our method for design consistency (COAST) starts from a correct solution (as defined by the user). Then, whenever the user modifies the constraints, COAST uses interval analysis techniques to deterministically follow the trajectory of the desired solution as it moves from the original solution to the new solution. In this section, the theory of using solution trajectories for design consistency is developed and the algorithm is outlined.

In variational CAD systems, we start with a fully constrained system of equations that has one or more possible solutions:

$$\underline{f}(\underline{d}_0) = \underline{b}_0 \Rightarrow \underline{d}_0 = \{x_1, x_2, \dots, x_i^*, \dots, x_{p_1}\},$$

There is no general approach for deciding which of the solutions is the correct one - this is based on the designer's intent. When the system of constraints is then changed by defining new values of the constraints, not only does the desired solution change, but the entire set of possible solutions changes:

$$\underline{f}(\underline{d}_1) = \underline{b}_1 \Rightarrow \underline{d}_1 = \{x'_1, x'_2, \dots, x'_j, \dots, x'_{p_2}\}.$$

Our goal then becomes to select the consistent solution out of the many possible ones. Our method also considers the case where the original system of constraints $\underline{f}_0(\underline{d}_0) = \underline{b}_0$ are transformed into a *new* system of algebraic equations $\underline{f}_1(\underline{d}_1) = \underline{b}_1$ as explained below.

14.5.1 DEFINITIONS IN DESIGN CONSISTENCY

Two n -dimensional systems of equations $\underline{f}(\underline{d}) = \underline{b}_0$ and $\underline{f}(\underline{d}) = \underline{b}_1$ ($\underline{b}_i \in \mathfrak{R}^n$) have the same number, p , of (possibly non-unique) solutions if \underline{d} is allowed to exist in the complex z -plane, C^n . In order to create a trajectory of the desired solution, the parameter vector is modified through a homotopy function,

$$\underline{H}(\underline{d}, t) = \underline{f}(\underline{d}) - \underline{b}_0 + t(\underline{b}_0 - \underline{b}_1) \quad (1)$$

When the original system of constraints $\underline{f}_0(\underline{d}_0) = \underline{b}_0$ is transformed into a *new* system of algebraic equations $\underline{f}_1(\underline{d}_1) = \underline{b}_1$, the following procedure is followed. We setup a convex homotopy between two systems of constraints, $\underline{f}_0(\underline{d}_0) = \underline{b}_0$ and $\underline{f}_1(\underline{d}_1) = \underline{b}_1$ as $\underline{H}'(\underline{d}, t) = (1-t)(\underline{f}_0(\underline{d}) - \underline{b}_0) + t(\underline{f}_1(\underline{d}) - \underline{b}_1)$, where t is the homotopy variable and varies from 0 to 1. Since the dimensions do not change

between design iterations (only their values), we can simplify $H'(\underline{d}, t)$ by noting that we can always construct a third system of equations, $f_{-1}(\underline{d}_0) = \underline{b}'_0$. Here, \underline{b}'_0 is calculated by substituting \underline{d}_0 into f_{-1} . Then we use the new system of equations to construct the desired homotopy, $H(\underline{d}, t) = (1-t)(f_{-1}(\underline{d}) - \underline{b}'_0) + t(f_{-1}(\underline{d}) - \underline{b}_1)$. It can easily be seen that $H'(\underline{d}, t)$ and $H(\underline{d}, t)$ have the same endpoints as t varies from 0 to 1.

Simplifying $H(\underline{d}, t)$,

$$H(\underline{d}, t) = (f_{-1}(\underline{d}) - \underline{b}'_0) - t(f_{-1}(\underline{d}) - \underline{b}'_0) + t(f_{-1}(\underline{d}) - \underline{b}_1) \Rightarrow$$

$$H(\underline{d}, t) = f_{-1}(\underline{d}) - \underline{b}'_0 + t(\underline{b}'_0 - \underline{b}_1).$$

Definition 14.2: Homotopy Function

The continuous transformation of the constraint-value vector from \underline{b}_0 to \underline{b}_1 is defined by a real-valued homotopy function, $B(t, \underline{b}_0, \underline{b}_1)$, with the following properties:

- $B(0, \underline{b}_0, \underline{b}_1) = \underline{b}_0$,
- $B(1, \underline{b}_0, \underline{b}_1) = \underline{b}_1$, and
- $B(t, \underline{b}_0, \underline{b}_1)$ is continuous for $t \in [0, 1]$

COAST uses the linear convex homotopy function, $B(t, \underline{b}_0, \underline{b}_1) = \underline{b}_0 + t \cdot (\underline{b}_1 - \underline{b}_0)$. Situations when different homotopy functions can be useful are discussed later in this chapter.

Every change in the constraint-value vector, \underline{b} , causes a change in the solution vector, \underline{d} . As the constraint-value vector is modified, the solution vector traces out a curve, called the *solution trajectory*.

Definition 14.3: Solution trajectory

A *solution trajectory* is the curve, $D(t, \underline{f}, B)$, that results from modifying the constraint-values according to the homotopy function, $B(t, \underline{b}_0, \underline{b}_1)$. The curve is denoted as $D(t, \underline{f}, B) = \{ \underline{f}(\underline{d}) = B(t, \underline{b}_0, \underline{b}_1) \}$ where $0 \leq t \leq 1$ and $\underline{f}: C^n \rightarrow \mathfrak{R}^n$ and has the following properties:

- $D(0, \underline{f}, B) = \underline{d}_0$,

- $D(1, \underline{f}, B) = \underline{d}_1$ = the new solution, and
- $D(t, \underline{f}, B)$ is continuous for $t \in [0, 1]$.

There are typically many solution trajectories for a given system of equations and a change in \underline{b} . However, since we are working with the trajectories between solutions, we can introduce the idea of the consistency of two solutions (in a strong and weak sense).

Definition 14.4: Consistent (strong)

Two solution vectors, \underline{d}_0 and \underline{d}_1 , are said to be *strongly consistent* if:

There exists a homotopy function, $B'(t, \underline{b}_0, \underline{b}_1)$, such that the corresponding solution trajectory, $D'(t, \underline{f}, B')$, has its endpoints at \underline{d}_0 and \underline{d}_1 , and

For every value of $t' \in [0, 1]$, there exists a neighborhood, ε , such that ε is above some tolerance level, ε_{tol} , and the interval, $[\underline{z}] = [D'(t', \underline{f}, B') - \varepsilon, D'(t', \underline{f}, B') + \varepsilon]$, contains exactly one solution.

Definition 14.5: Consistent (weak)

Two solution vectors, \underline{d}_0 and \underline{d}_1 , are said to be *weakly consistent* if:

There exists a homotopy function, $B'(t, \underline{b}_0, \underline{b}_1)$, such that the corresponding solution trajectory, $D'(t, \underline{f}, B')$, has its endpoints at \underline{d}_0 and \underline{d}_1 , and

For some value of $t' \in [0, 1]$, there exists a neighborhood, $\varepsilon = \varepsilon_{tol}$ such that the interval, $[\underline{z}] = [D'(t', \underline{f}, B') - \varepsilon, D'(t', \underline{f}, B') + \varepsilon]$, contains more than one solution.

Definition 14.6: Inconsistent

Two solution vectors, \underline{d}_0 and \underline{d}_1 , are said to be *inconsistent* if:

There is no homotopy function, $B'(t, \underline{b}_0, \underline{b}_1)$, such that the corresponding solution trajectory, $D'(t, \underline{f}, B')$, has its endpoints at \underline{d}_0 and \underline{d}_1 .

In order for two solutions to be strongly consistent, the solution trajectory between them cannot intersect any other solution trajectory. If two solution trajectories do intersect, then it is ambiguous which one to follow after the intersection point. While one can use heuristics to move through an ambiguous area in what is hopefully a consistent manner, it can never be guaranteed.

14.5.2 THEOREMS IN DESIGN CONSISTENCY

Current methods for solving systems of nonlinear equations in complex analysis are restricted to solving systems of polynomial equations. As we require COAST to work with more general systems of nonlinear equations, we are restricted to the real plane where more widely applicable techniques such as Newton-Raphson are available. The previous definitions for trajectories and consistency remain the same with the understanding that the solution trajectories between solutions must remain in the real plane (\underline{d} is restricted to the reals: $\underline{f}(\underline{d}) = \underline{b}$, $\underline{f}: \mathfrak{R}^n \rightarrow \mathfrak{R}^n$). However, in the real plane, we are not guaranteed a solution. To check for the existence and uniqueness of solutions within a given interval, we will use a theorem from interval analysis (see Appendix A for a review of interval analysis techniques):

Theorem 14.1:

Let $\underline{f}: \mathfrak{R}^n \rightarrow \mathfrak{R}^n$ be continuously differentiable over the domain of interest. If $\underline{d} \in [\underline{z}]$ (where $[\underline{z}]$ is some real interval vector) and $[\underline{d}]$ denotes the Gauss-Seidel operator, then:

Every zero $\underline{d}^* \in [\underline{z}]$ of \underline{f} satisfies $\underline{d}^* \in [\underline{d}]$.

If $[\underline{d}] \cap [\underline{z}] = \emptyset$ then \underline{f} contains no zero in x .

If $\underline{d} \in \text{int}([\underline{z}])$ and $0 \neq [\underline{d}] \subseteq \text{int}([\underline{z}])$ then \underline{f} contains a unique zero in x .

The proof is given in [16]. The Gauss-Seidel operator is discussed in [8, 16].

In order to show that two solutions are strongly consistent, we prove the following theorem. The general concept is to draw a region around the entire solution trajectory and if no other solution trajectory passes through the region, then the two solutions must be strongly consistent.

Theorem 14.2 (sufficient condition for strong consistency):

Let \underline{d}_0 and \underline{d}_1 be two solution vectors. Let $B'(t, \underline{b}_0, \underline{b}_1)$ be a parameter trajectory function such that the corresponding solution trajectory, $D'(t, \underline{f}, B')$, has its endpoints at \underline{d}_0 and \underline{d}_1 . Let $[\underline{d}]$ be a region that encloses the solution trajectory and let $[\underline{z}] = [\text{inf}([\underline{d}]) - \delta, \text{sup}([\underline{d}]) + \delta]$ be the region just surrounding $[\underline{d}]$. If $[\underline{d}] \subseteq \text{int}([\underline{z}])$ for every $t \in [0, 1]$ then \underline{d}_0 and \underline{d}_1 are strongly consistent.

Proof:

There are three cases: \underline{d}_0 and \underline{d}_1 are either strongly consistent, weakly consistent, or inconsistent. Since there is a solution trajectory between \underline{d}_0 and \underline{d}_1 , they can only

be either strongly consistent or weakly consistent. Assume that \underline{d}_0 and \underline{d}_1 are weakly consistent. For some value of $t = t^* \in [0,1]$, there exists a neighborhood, $\varepsilon = \varepsilon_{tol}$, such that the interval, $[\underline{z}'] = [D(t^*, \underline{f}, B') - \varepsilon, D(t^*, \underline{f}, B') + \varepsilon]$, contains more than one solution. However, it is given that the interval $[\underline{z}] = [\inf([\underline{d}]) - \delta, \sup([\underline{d}]) + \delta]$ only has one solution (from Theorem 14.1). Since $[\underline{z}'] \subseteq [\underline{z}]$, $[\underline{z}']$ must have only one solution as well which leads us to a contradiction and to the conclusion that \underline{d}_0 and \underline{d}_1 are strongly consistent. ■

Rather than relying on a unique solution in such a large region, we can break down the solution trajectory into separate regions, testing the solution consistency in each region separately. The algorithm introduced later will use this concept extensively.

Theorem 14.3 (transitive theorem of strong solution consistency):

Let \underline{d}_0 , \underline{d}^* , and \underline{d}_1 be three solutions on one solution trajectory, $D'(t, \underline{f}, B)$, such that $D'(0, \underline{f}, B) = \underline{d}_0$, $D'(1, \underline{f}, B) = \underline{d}_1$, and $D'(t^*, \underline{f}, B) = \underline{d}^*$ (where $0 < t^* < 1$). If \underline{d}_0 and \underline{d}^* are strongly consistent and \underline{d}^* and \underline{d}_1 are strongly consistent, then \underline{d}_0 and \underline{d}_1 are strongly consistent.

Proof:

Since we are given the existence of a homotopy function, $B'(t, \underline{b}^0, \underline{b}^1)$, such that the corresponding solution trajectory, $D'(t, \underline{f}, B')$, has its endpoints at \underline{d}_0 and \underline{d}_1 , we only have to prove that for every value of $t' \in [0,1]$, there exists a neighborhood, $\varepsilon \geq \varepsilon_{tol}$, such that the interval, $[\underline{z}] = [D'(t', \underline{f}, B') - \varepsilon, D'(t', \underline{f}, B') + \varepsilon]$, contains exactly one solution. But this must be the case since the interval $[\underline{z}]$ must exist from \underline{d}_0 to \underline{d}^* and from \underline{d}^* to \underline{d}_1 (since they are strongly consistent) which covers the entire solution trajectory so \underline{d}_0 and \underline{d}_1 are therefore strongly consistent. ■

Finally, we prove that design consistency is a unique relationship. After each design step, every solution has exactly one consistent solution that corresponds to it.

Theorem 14.4 (uniqueness theorem of solution consistency):

Let \underline{d}_0 , \underline{d}^* , and \underline{d}_1 be three solution vectors: \underline{d}_0 is the solution to $\underline{f}_0(\underline{d}) = \underline{b}_0$ and \underline{d}^* , and \underline{d}_1 are solutions to $\underline{f}_1(\underline{d}) = \underline{b}_1$. If \underline{d}_0 and \underline{d}^* are strongly consistent and

\underline{d}_0 and \underline{d}_1 are strongly consistent, then $\underline{d}^* = \underline{d}_1$.

Proof:

Assume $\underline{d}^* \neq \underline{d}_1$. Since \underline{d}_0 and \underline{d}^* are strongly consistent, there is a solution trajectory from \underline{d}_0 to \underline{d}^* . Likewise, there is a solution trajectory from \underline{d}_0 and \underline{d}_1 . Since the two trajectories start from the same point and end at different points, the two trajectories have to split at some point, t^* . However, after they split, there will be some value of $\varepsilon \geq \varepsilon_{tol}$ such that the interval, $[\underline{z}] = [D'(t^*, \underline{f}, B') - \varepsilon, D'(t^*, \underline{f}, B') + \varepsilon]$, contains exactly two solutions. But since \underline{d}_0 and \underline{d}^* are strongly consistent, this is a contradiction which leads us to the conclusion that $\underline{d}^* = \underline{d}_1$. ■

14.6 COAST ALGORITHM FOR DESIGN CONSISTENCY

14.6.1 MEAN VALUE THEOREM

Rather than using approximate predictor-corrector algorithms to follow a solution trajectory, COAST uses interval analysis techniques to create guaranteed bounds for enclosures of a solution to a system of equations subject to change (see Appendix A for a review of interval analysis techniques). COAST iteratively solves the system of equations, $\underline{H}(\underline{d}, t) = 0$ when t varies from t^* to $(t^* + \Delta t)$ as (t varies from 0 to 1):

$$\underline{H}(\underline{d}_0, t^*) = \underline{f}(\underline{d}_0) - \underline{b}_0 + t^*(\underline{b}_0 - \underline{b}_1) \Rightarrow \tag{2}$$

$$\underline{f}(\underline{d}_0) = \underline{b}_0 - t^*(\underline{b}_0 - \underline{b}_1)$$

$$\underline{H}(\underline{d}_1, t^* + \Delta t) = \underline{f}(\underline{d}_1) - \underline{b}_0 + (t^* + \Delta t)(\underline{b}_0 - \underline{b}_1) \Rightarrow \tag{3}$$

$$\underline{f}(\underline{d}_1) = \underline{b}_0 - (t^* + \Delta t)(\underline{b}_0 - \underline{b}_1)$$

The interval extension of the mean-value theorem [11] guarantees the existence of an interval vector, $[\underline{z}]$, that exhibits the following property:

$$\underline{f}(\underline{d}_1) - \underline{f}(\underline{d}_0) \in \underline{f}'([\underline{z}])(\underline{d}_1 - \underline{d}_0) \tag{4}$$

where $\underline{d}_0, \underline{d}_1 \in [\underline{z}]$.

Let $\underline{b}_0^* = \underline{b}_0 - t^*(\underline{b}_0 - \underline{b}_1)$ and $\underline{b}_1^* = \underline{b}_0 - (t^* + \Delta t)(\underline{b}_0 - \underline{b}_1)$. Substituting for $\underline{f}(\underline{d}_1)$ and $\underline{f}(\underline{d}_0)$ yields:

$$\underline{b}_1^* - \underline{b}_0^* \in \underline{f}'([\underline{z}]) (\underline{d}_1 - \underline{d}_0) \quad (5)$$

Which can be represented as a linear interval system of the form $\underline{A}\underline{d} = \underline{b}$ where:

$$\underline{A} = \underline{f}'([\underline{z}]), \underline{d} = \underline{d}_1 - \underline{d}_0, \text{ and } \underline{b} = \underline{b}_1^* - \underline{b}_0^* \quad (6)$$

The solution to this system, $\underline{d} = \underline{A}^H \underline{b}$ (defined as the hull of the solution set of the linear interval system), is enclosed by $\underline{d} = \underline{A}^{-1} \underline{b}$, where \underline{A}^{-1} is the interval extension of the matrix inverse of \underline{A} (which is computable if \underline{A} is regular).

$$\underline{d} = \underline{d}_1 - \underline{d}_0 \supseteq \underline{A}^{-1} \underline{b} \quad (7)$$

$$\underline{d}_1 \supseteq \underline{d}_0 + \underline{A}^{-1} \underline{b} \quad (8)$$

$$\underline{d}_1 \supseteq \underline{d}_0 + \underline{f}'([\underline{z}])^{-1} (\underline{b}_1^* - \underline{b}_0^*) \quad (9)$$

Equation (9) gives an initial guaranteed enclosure for \underline{d}_1 . The bounds can be further tightened using iterative interval solution techniques (e.g., Gauss-Seidel iteration).

14.6.2 RIGOROUS SENSITIVITY ANALYSIS ALGORITHM

Using equation (9) and techniques from [8, 15, 16], we can create a simple algorithm for following the homotopy curve, $\underline{H}(\underline{d}, t) = \underline{f}'(\underline{d}) - \underline{b}_0 + t(\underline{b}_0 - \underline{b}_1)$ as t varies from 0 to 1 (see Figure 14.5).

There are three areas in which researchers have enhanced this basic rigorous sensitivity analysis algorithm. First, there are two obvious qualitative steps used in the algorithm: "Create $[\underline{z}]$ " and "Increase/Decrease t ". These two parameters, $[\underline{z}]$ and t , define an $n+1$ dimensional box around the current solution in which the next solution must lie. If the box is too small, the convergence will be slowed down greatly. If the box is too large, however, multiple solution trajectories may enter it, requiring COAST to repeat the process with a smaller box. To better quantify the creation of $[\underline{z}]$ and t , Kearfott and Xing [9] have developed an interval step control method to select an optimal box. A less obvious area for improvement is in the use of the interval extension of the Jacobian. A *slope* is an interval extension of the gradient of a function and the Jacobian is merely one such slope. A better estimate can be obtained by using more accurate slopes [e.g., 11]. It should be understood that due to the rigorous nature of interval analysis, these improvements do not affect the robustness of the algorithm - only the speed at which it converges.

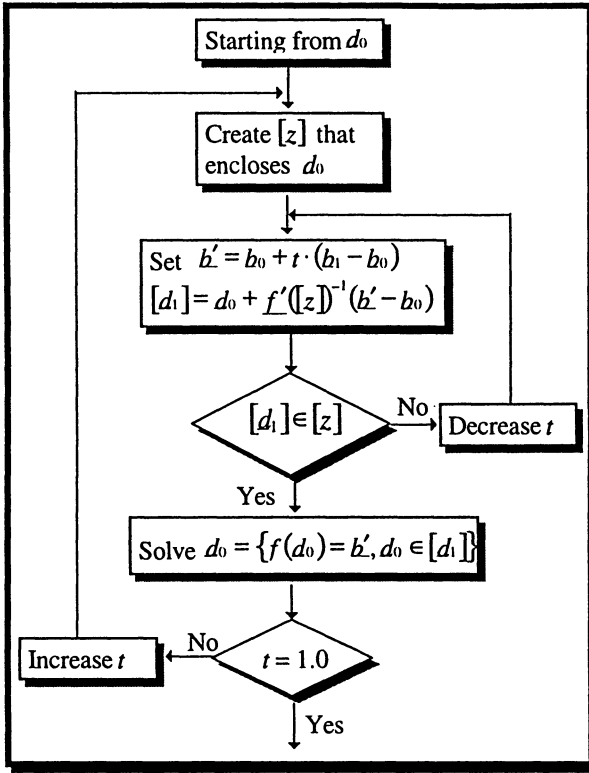


Figure 14.5 COAST Outline

It is now proven that each successful iteration of the algorithm guarantees strong consistency (see Definition 14.4).

Theorem 14.5:

If the following conditions hold:

1. $[d_1] = d_0 + f'([z])^{-1}(b' - b_0)$, where $b' = b_0 + t \cdot (b_1 - b_0)$,
2. $d_0 \in [d_1] \subset [z]$,
3. $d'_0 = \{f(d'_0) = b', d'_0 \in [d_1]\}$, and
4. d'_0 is unique,

then d_0 and d'_0 are strongly consistent.

Proof:

The two solutions, d_0 and d'_0 , are bounded by the interval, $[z]$ and are the

endpoints of the solution trajectory as t is changed from t_0 to t' . Since $[d_1]$ is a proper subset of $[z]$, there is a neighborhood about $[d_1]$, $([d_1] + [-\delta, \delta])$, which is also a proper subset of $[z]$. Because d'_0 is a unique solution to $f(d'_0) = b'$, we know that $[d] \subseteq \text{int}([z])$ and thus, d'_0 is strongly consistent to d_0 by Theorem 14.2. ■

From Theorem 14.4, we know that d'_0 is the unique consistent solution. From Theorem 14.3, we see that we can incrementally move t from 0 to 1, maintaining a strongly consistent solution at each increment, and the final solution will be strongly consistent with the original solution.

14.6.3 BIFURCATIONS AND INFEASIBLE REGIONS

When following a single solution trajectory, two types of problems can arise: bifurcations (when one trajectory splits into two trajectories or two trajectories intersect - see Figure 14.6-A) and infeasible regions (when there is no real value for the desire system of constraints for some value of the homotopy variable - see Figure 14.6-B).

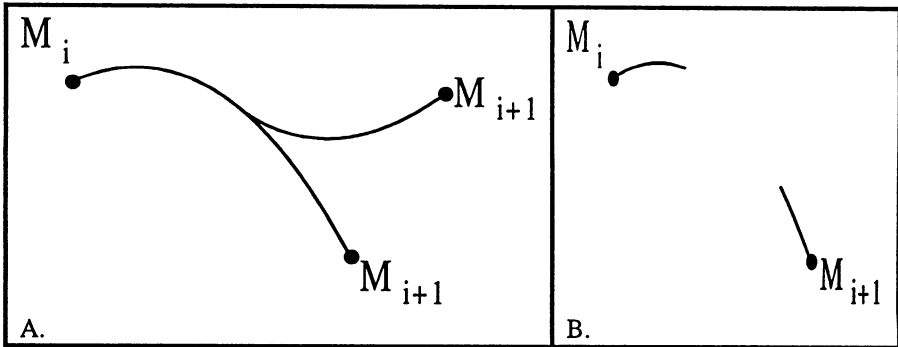


Figure 14.6 Bifurcations and Infeasible Regions

Bifurcations

The classic approach to handling bifurcations is to perturb the coefficients of the homotopy. Doing so creates a new system of constraints which (hopefully) doesn't have the same bifurcation. The path taken, however, is dependent on how the system is perturbed. Although it seems likely that a solution that lies on one of the bifurcation paths would be "more consistent" than a solution that does not, we

maintain our original assumption that only one solution is acceptable to the designer. Any bifurcation introduces ambiguity into the consistency of the design. We assert that the handling of bifurcations has to be domain specific, using the exact interpretations of the dimensions to help decide which path is more desirable.

Infeasible Regions

It is possible that the homotopy would become infeasible for some value of t . Due to our simplified form of the convex homotopy (where we only modify the constraint values), we know that the number of complex solutions to the system of equations does not change through the homotopy. Also, since complex solutions come in conjugate pairs, we can infer the following: (1) the start of any infeasible region among the reals is actually a bifurcation if the dimension values are allowed to be complex numbers, and (2) the system of equations at the bifurcation point has multiplicity of at least two. Any bifurcation routines that can be extended to the complex plane can be used to traverse infeasible regions. We know of algorithms for finding complex roots of polynomial systems (in which case the COAST algorithm can be extended). If it is not possible to find complex roots, then an alternative approach may be to try different homotopies in hopes of moving the solution trajectory around the infeasible region.

14.7 DESIGN OF CANTILEVER BEAM

This demonstration is that of a cantilever beam for a uniformly distributed load. To make the design process easier, a constraint model is constructed that defines the dimensions and attributes. The constraint model for the beam is given in Appendix B.

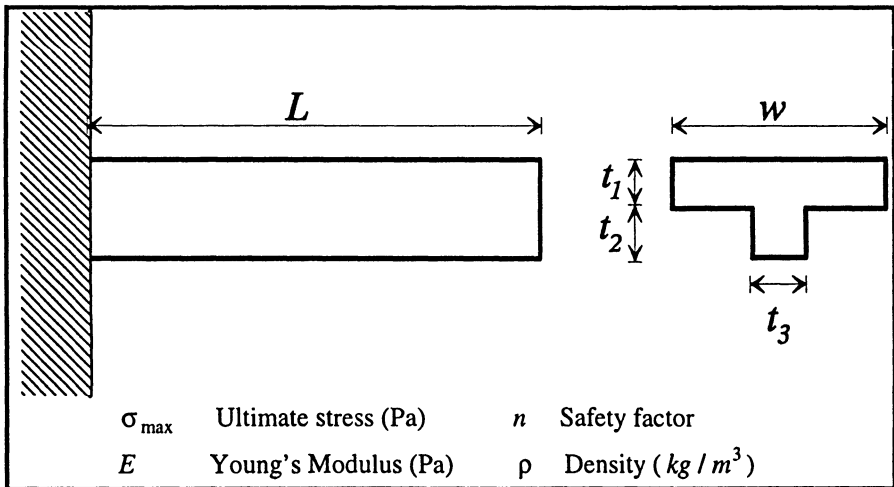


Figure 14.7 Dimensions of a 'T'-Shaped Cantilever Beam

Additionally, the following attributes are defined for the beam in the constraint model:

Volume, V	Mass, M
Total Surface Area, S	Maximum Moment, M_{\max}
Maximum Load, p_{\max}	Maximum Allowable Load, p_{allow}
End Angle at p_{\max} , θ_{\max}	End Angle at p_{allow} , θ_{allow}
End Deflection at p_{\max} , Δ_{\max}	End Deflection at p_{allow} , Δ_{allow}

14.7.1 DESIGN EXECUTION

For this problem, the requirements for a satisfactory design, θ_0^Q , are:

- $L > 20$ cm
- $w > 2.0$ cm
- $M < 1.5$ kg
- $p_{\text{allow}} > 2000$ kg
- $\Delta_{\text{allow}} < 0.1$ mm
- material is steel

Consider the following design execution to satisfy those requirements:

$$\theta_0^C: \quad M_0:$$

$$\left\{ \begin{array}{l} t_1 = 0.01 \text{ (m)} \\ t_2 = 0.015 \text{ (m)} \\ t_3 = 0.01 \text{ (m)} \\ w = 0.02 \text{ (m)} \\ L = 0.20 \text{ (m)} \\ \sigma_{\max} = 385 \cdot 10^6 \text{ (Pa)} \\ \rho = 7850 \text{ (kg} \cdot \text{m}^{-3}\text{)} \\ n = 2.5 \\ E = 200000 \cdot 10^6 \end{array} \right. \rightarrow \left\{ \begin{array}{l} t_1 = 0.01 \text{ (m)} \\ t_2 = 0.015 \text{ (m)} \\ t_3 = 0.01 \text{ (m)} \\ w = 0.02 \text{ (m)} \\ L = 0.20 \text{ (m)} \\ \sigma_{\max} = 385 \cdot 10^6 \text{ (Pa)} \\ \rho = 7850 \text{ (kg} \cdot \text{m}^{-3}\text{)} \\ n = 2.5 \\ E = 200000 \cdot 10^6 \end{array} \right.$$

$\Delta\theta_0:$

$$\left\{ \begin{array}{l} p_{\text{allow}} = 2000 \text{ (kg)} \\ \Delta_{\text{allow}} = 0.0001 \text{ (m)} \end{array} \right.$$

⇓

$$\begin{array}{l}
 \theta_1^C: \\
 \left\{ \begin{array}{l}
 t_1 = 0.01 (m) \\
 t_2 = 0.015 (m) \\
 t_3 = 0.01 (m) \\
 p_{allow} = 2000 (kg) \\
 \Delta_{allow} = 0.0001 (m) \\
 \sigma_{max} = 385 \cdot 10^6 (Pa) \\
 \rho = 7850 (kg \cdot m^{-3}) \\
 n = 2.5 \\
 E = 200000 \cdot 10^6
 \end{array} \right. \rightarrow \begin{array}{l}
 M_1: \\
 \left\{ \begin{array}{l}
 t_1 = 0.01 (m) \\
 t_2 = 0.015 (m) \\
 t_3 = 0.01 (m) \\
 w = 0.0116 (m) \\
 L = 0.18 (m) \\
 \sigma_{max} = 385 \cdot 10^6 (Pa) \\
 \rho = 7850 (kg \cdot m^{-3}) \\
 n = 2.5 \\
 E = 200000 \cdot 10^6
 \end{array} \right.
 \end{array}$$

$$\Delta\theta_1: \left\{ \begin{array}{l}
 w = 0.04 (m) \\
 L = 0.20 (m) \\
 M = 1.3 (kg)
 \end{array} \right.$$

↓

$$\begin{array}{l}
 \theta_2^C: \quad M_2: \\
 \left\{ \begin{array}{l}
 t_1 = 0.01(m) \\
 w = 0.04(m) \\
 L = 0.20(m) \\
 M = 1.3(kg) \\
 \Delta_{allow} = 0.0001(m) \\
 \sigma_{max} = 385 \cdot 10^6(Pa) \\
 \rho = 7850(kg \cdot m^{-3}) \\
 n = 2.5 \\
 E = 200000 \cdot 10^6
 \end{array} \right. \rightarrow \left\{ \begin{array}{l}
 t_1 = 0.01(m) \\
 t_2 = 0.0195(m) \\
 t_3 = 0.0219(m) \\
 w = 0.04(m) \\
 L = 0.20(m) \\
 \sigma_{max} = 385 \cdot 10^6(Pa) \\
 \rho = 7850(kg \cdot m^{-3}) \\
 n = 2.5 \\
 E = 200000 \cdot 10^6
 \end{array} \right.
 \end{array}$$

$$\Delta\theta_2: \{\emptyset\}$$

$\langle \theta_0^C, M_0 \rangle$ is the initial solution by the designer of a satisfactory configuration yielding the following attribute values:

$V = 0.00007$	$M = 0.5495$	$P_{max} = 3649.39$	$\theta_{max} = 0.0013613$	$\Delta_{max} = 0.000204196$
$S = 0.1835$	$M_{max} = 72.9878$	$P_{allow} = 1459.76$	$\theta_{allow} = .000544522$	$\Delta_{allow} = .0000816783$

The changes in the constraint system, $\Delta\theta_0^C$, are motivated by the observation that p_{allow} should be at least 2000 and that Δ_{allow} could be eased to 0.0001. In order to maintain a fully constrained system, w and L are unconstrained.

After replacing the constraints, all but two of the part's dimensions (w and L) are directly constrained. Mathematically, θ_1^C has four possible solutions:

- 1. $M_1(w, L): w = 0.01158303 \quad L = 0.18290615$
- 2. $M_1(w, L): w = 0.01158303 \quad L = -0.18290615$
- 3. $M_1(w, L): w = -0.26639935 \quad L = 0.144491157$
- 4. $M_1(w, L): w = -0.26639935 \quad L = -0.144491157$

The homotopy curves of these four solutions are shown in Figure 14.8. Of the four solutions, only the first one is possible so that is the one intended yielding the following configuration:

$M_1:$

$t_1 = 0.01$	$w = 0.01158303$	$\rho = 7850$
$t_2 = 0.015$	$L = 0.18290615$	$n = 2.5$
$t_3 = 0.01$	$\sigma_{max} = 385 \cdot 10^6$	$E = 200000 \cdot 10^6$

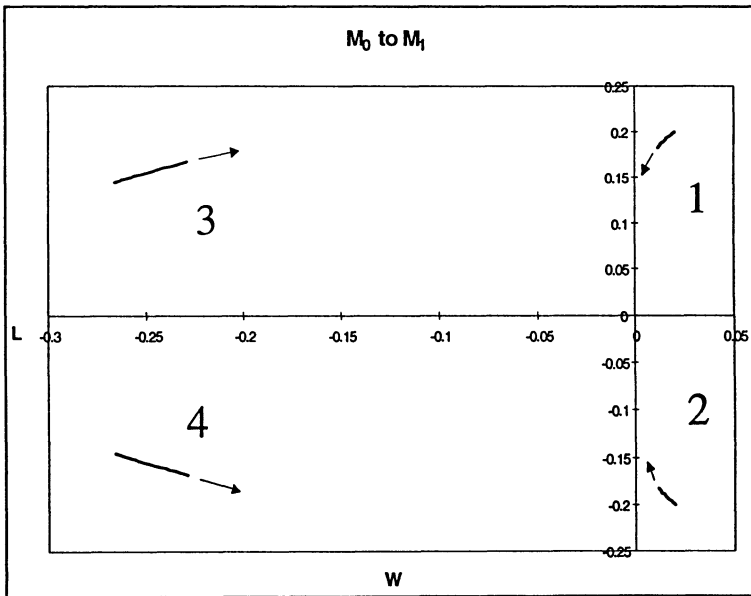


Figure 14.8 Solution Trajectories for M_1

Calculating the new values of the attributes:

V = 0.000049	M = 0.382	p_{max} = 4935.06	θ_{max} = 0.00179876	Δ_{max} = 0.000246753
S = 0.136	M_{max} = 82.5505	p_{allow} = 2000.00	θ_{allow} = 0.000719504	Δ_{allow} = 0.0001

It is then observed that while p_{allow} is adequate, the width and length are too small now. A new iteration is then performed, constraining the width, length, mass, and the deflection at the maximum load.

Again, all but two dimensions (t_3 and t_2) are directly constrained. There are three possible solutions to this system of equations (including one impractical solution):

1. $M_2(t_2, t_3)$: $t_2 = -0.001709$ $t_3 = -0.25041$
2. $M_2(t_2, t_3)$: $t_2 = 0.019505$ $t_3 = 0.02194$
3. $M_2(t_2, t_3)$: $t_2 = 0.009508$ $t_3 = 0.04501$

When M_1 was calculated, if all solutions could have been identified, then the correct solution could be deduced by simple deduction (only one solution was practical). In this case, however, there is no simple way to deduce which solution is the correct one. While the first solution can be deduced to be impractical, the other two solutions are both reasonable and under different circumstances, either one could be the desired solution. In order to compute which solution is the correct one, the homotopy is created and the desired solution trajectory is followed. The solution trajectories for the three possible solutions are shown in Figure 14.9. By using the algorithm from Section 14.3, The COAST method is able to follow the solution trajectory of the desired solution as it moves to its new, correct configuration:

M_2 :

$t_1 = 0.01$	$w = 0.04$	$\rho = 7850$
$t_2 = 0.019505$	$L = 0.20$	$n = 2.5$
$t_3 = 0.021943$	$\sigma_{max} = 385 \cdot 10^6$	$E = 200000 \cdot 10^6$

The calculated attributes yield:

V = 0.0001656	M = 1.3	p_{max} = 15273.9	θ_{max} = 0.00164497	Δ_{max} = 0.000246746
S = 0.02863	M_{max} = 305.479	p_{allow} = 6109.58	θ_{allow} = 0.000657988	Δ_{allow} = 0.0001

Since M_2 satisfies θ_2^0 (the requirements were never modified so $\theta_2^0 = \theta_1^0 = \theta_0^0$), the design process is completed.

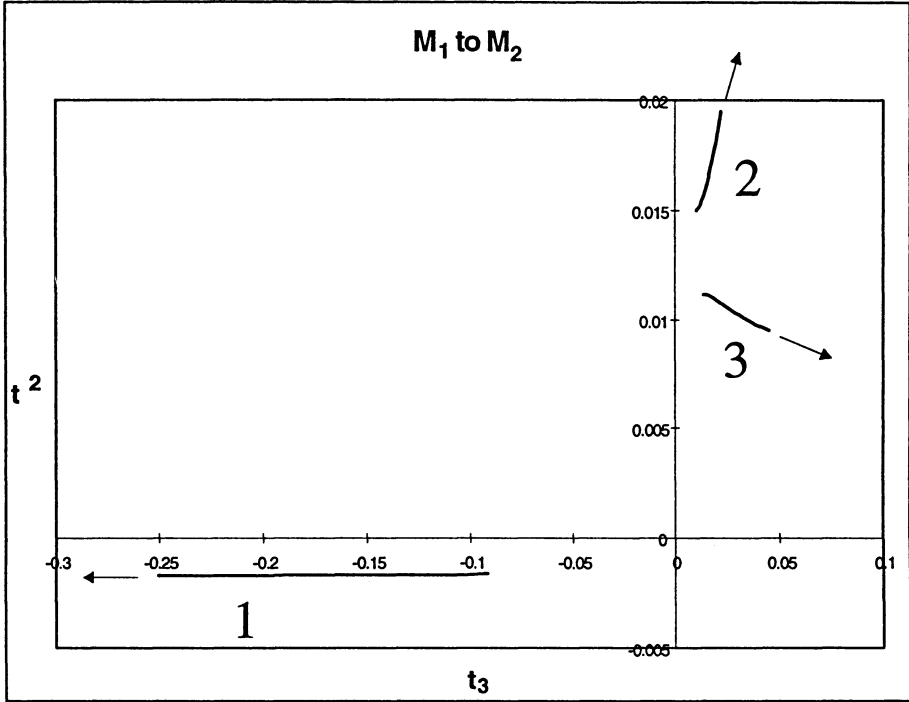


Figure 14.9 Solution Trajectories for M_2

14.7.2 COMPARISON WITH OTHER METHODS

For a comparison of other methods, we use Newton’s method to solve the same system of constraints, θ_2^C , (using the FindRoot command in Mathematica). The initial value of each dimension is taken to be either the value directly constrained by the user or, if it does not exist, the value of the dimension from the previous design iteration. The following table shows the initial values of each dimension and the value that it converges to. As can be seen from the values of t_2 and t_3 , Newton’s method converges to the wrong solution.

Table 14.1 Summary Results for Newton’s Method

Dimension	Initial Value	Converged Value	Dimension	Initial Value	Converged Value	Dimension	Initial Value	Converged Value
t_1	0.01	0.01	w	0.04	0.04	E	$2 \cdot 10^{11}$	$2 \cdot 10^{11}$
t_2	0.015	0.0095	L	0.2	0.2	ρ	7850	7850
t_3	0.01	0.04502	n	2.5	2.5	σ_{\max}	$3.85 \cdot 10^8$	$3.85 \cdot 10^8$

In order to try to converge to the correct solution, fixed step continuation (as reviewed in Section 14.2) is attempted and the number of steps is increased until correct re-convergence is attained. The following table shows the results.

Table 14.2 Fixed Point Continuation Comparison

	Dimension	Converged Value
Number Steps = 2	t_1	0.01
	t_2	0.0095
	t_3	0.04502
	w	0.04
	L	0.2
	n	2.5
	E	$2 \cdot 10^{11}$
	ρ	7850
	σ_{\max}	$3.85 \cdot 10^8$
Number Steps = 3	t_1	0.01
	t_2	0.01951
	t_3	0.02194
	w	0.04
	L	0.2
	n	2.5
	E	$2 \cdot 10^{11}$
	ρ	7850
	σ_{\max}	$3.85 \cdot 10^8$

By increasing the number of steps, Newton’s method is able to converge to the correct solution. Finally, we tried reducing the speed of the convergence by modifying the relaxation factor. This was done using the DampingFactor option in Mathematica. After trying different values between 1 and 0.001, there was no change in the converged solution.

In conclusion, while Newton’s method was able to converge to the correct

solution, it succeeded or failed depending on the number of steps taken. For this system of constraints, it failed for one or two steps and it succeeded for three or more steps. These numbers would, of course, be different for any other system of constraints, requiring user intervention to select a proper number of iterations. Setting the number of iterations to an arbitrarily large number would, indeed, probably work for a large number of design cases, but it would be prohibitively time-consuming and one would never know if it failed without monitoring the convergence. Many of the more advanced continuation methods mentioned in Section 14.2 would be able to handle this problem. But the designer could never be sure that the solver converged to the right solution. The COAST method will either guarantee a consistent solution or it will recognize those situations in which it cannot do so (bifurcations or infeasible regions).

14.8 SUMMARY

Variational design is a powerful paradigm for design, but has been limited in its application by the inability to characterize multiple solutions resulting from systems of nonlinear equations. Previous methods have required the user to either (1) manually guide the constraint solver routines, (2) manually select the correct solution from a list, or (3) input enough extra information so as there is no ambiguity. Additionally, these methods are not able to guarantee a consistent design. As opposed to these methods, we have used an interval continuation method to deterministically trace a convex homotopy between the original and new systems of constraints. An algorithm was presented and demonstrated to automatically reconverge to a consistent solution.

The applicability of design consistency methods to different design areas is presented in chapter 20. The interval-based continuation approach developed in this chapter is implemented in the next two chapters to constraint-based curve design (Chapter 15) and 3-D shoe design (Chapter 16). Constraint-based design of faired parametric curves is demonstrated in Chapter 15 when the constraints are on the whole curve (e.g., a curve having a certain arc length or being a given distance from another graphical object) and not solely on the defining points of the curve. Because the constraints will often be changed over time, it is shown that finding the global optimum of the fairing objective function is often less important than finding a consistent solution and that a local optimum of the objective function should at least be an option in such a system. Necessary conditions of the optimization problem are used to locate the desired local optimum. As there are normally many local optimums, the COAST methodology is applied to maintain a consistent solution. Besides demonstrating this technique with relations between Bezier curves, an example is shown from apparel design. Finally, as a proof-of-concept, a 3-D shoe design system is demonstrated in Chapter 16 that can grade an upper design to any size by modifying the measurements of the parametrized model of a foot. The system is then able to create the corresponding manufacturable patterns and render the design.

Future work in this area involves increasing the robustness of the COAST method and applying it to different areas. To make it more robust involves incorporating heuristics and more intelligent mathematical techniques to handle cases of intersecting trajectories or infeasible regions. One of the areas we are applying COAST is actually in constraint-based curve design. Constraints on curves are represented as optimization problems. By looking at the necessary conditions and we can reduce the optimization problem to a system of nonlinear equations and use COAST to maintain a consistent solution.

APPENDIX A - INTERVAL ANALYSIS TECHNIQUES

This appendix gives a quick overview of interval analysis techniques. More thorough coverage of interval analysis techniques is available in [16, 8]. The field of interval analysis has come up with many powerful methods for analyzing systems of equations when they are subject to change. These include:

- Iteratively improving an enclosure of a solution to a system of equations
- Rigorous sensitivity analysis of a system of equations
- Existence and uniqueness tests of systems of equations over intervals of parameters

Let $[x]$ be an interval and let \tilde{x} be any real number in interval $[x]$,

$$[x] \equiv [\downarrow[x], \uparrow[x]] := \{ \tilde{x} \in \mathfrak{R} \mid \downarrow[x] \leq \tilde{x} \leq \uparrow[x] \}$$

Every interval, $[x]$, has a midpoint,

$$\hat{x} = \text{mid}([x]) = (\uparrow[x] + \downarrow[x]) / 2$$

and a radius,

$$\text{rad}([x]) = (\uparrow[x] - \downarrow[x]) / 2$$

The interior of an interval is denoted by

$$\text{int}([x]) = \{ \tilde{x} \in \mathfrak{R} \mid \downarrow[x] < \tilde{x} < \uparrow[x] \}$$

The hull of a nonempty bounded subset of \mathfrak{R} , S , is the tightest interval that encloses S and is denoted by

$$\text{Hull}\{S\} = [\text{inf}(S), \text{sup}(S)] .$$

Arithmetic operations are defined on intervals $[x]$ and $[y]$ as follows:

- $[x] \circ [y] = \text{Hull} \{ \downarrow[x] \circ \downarrow[y], \downarrow[x] \circ \uparrow[y], \uparrow[x] \circ \downarrow[y], \uparrow[x] \circ \uparrow[y] \}$ for $\circ \in \{+, -, *, \setminus\}$
- $\varphi([x]) = [\varphi(\downarrow[x]), \varphi(\uparrow[x])]$ for $\varphi \in \{\text{sqrt}, \text{exp}, \text{ln}\}$

- $\text{sqr}([x]) = [0, [x]^2]$ if $0 \in [x]$ or $[\downarrow[x]^2, \uparrow[x]^2]$ otherwise

Other interval operations can be defined similarly.

Interval vectors are defined as follows: let \mathfrak{R}^n denotes the vector space of (column) vectors with n real components. \mathcal{IR}^n denotes the set of *interval vectors* $[\underline{x}] = ([x_1], [x_2], \dots, [x_n])^T$ with n components $[x_1], [x_2], \dots, [x_n] \in \mathcal{IR}$. We interpret $[\underline{x}] \in \mathcal{IR}^n$ as the set of all vectors $\underline{x} \in \mathfrak{R}^n$ with $x_i \in [x_i]$ for $i=1, \dots, n$. For $n=2$, this set is a rectangle in the plane.

A.1 SOLVING SYSTEMS OF INTERVAL EQUATIONS

We wish to find a solution to the system of equations,

$$\underline{f}(\underline{x}) = \underline{0} \quad (1)$$

within a given interval, $\underline{x} \in [\underline{z}]$, $[\underline{z}] \in \mathcal{IR}^n$. Solution strategies can be adapted from the mean value theorem:

$$\underline{f}(\text{mid}([\underline{z}])) - \underline{f}(\underline{x}^*) \in \underline{f}'([\underline{z}]) \cdot (\text{mid}([\underline{z}]) - \underline{x}^*) \text{ for all } \underline{x}^* \in [\underline{z}] \quad (2)$$

Since \underline{x}^* is a zero of $\underline{f}(\underline{x}) = \underline{0}$,

$$\underline{f}(\text{mid}([\underline{z}])) \in \underline{f}'([\underline{z}]) \cdot (\text{mid}([\underline{z}]) - \underline{x}^*) \quad (3)$$

It is commonly known that iteration methods such as this one work better if preconditioned by a real matrix (typically, the inverse of the midpoint matrix of $\underline{f}'([\underline{z}])$).

$$R \cdot \underline{f}(\text{mid}([\underline{z}])) \in R \cdot \underline{f}'([\underline{z}]) \cdot (\text{mid}([\underline{z}]) - \underline{x}^*) \quad (4)$$

If we let $A = R \cdot \underline{f}'([\underline{z}])$ and $\underline{b} = R \cdot \underline{f}(\text{mid}([\underline{z}]))$ then (4) can be rewritten as

$$\underline{b} \in A \cdot (\text{mid}([\underline{z}]) - \underline{x}^*) \quad (5)$$

and iteratively solved using Gauss Seidel iteration. Let $\underline{c} \approx \text{mid}([\underline{z}]) \in [\underline{z}]$ and $\underline{\tilde{x}} \in [\underline{z}]$. The system is rewritten as $A \cdot (\underline{c} - \underline{\tilde{x}}) = \underline{b}$. Writing this system in terms of its individual components yields

$$\sum_{j=1}^n A_{ij} \cdot (c_j - \tilde{x}_j) = b_i \quad (6)$$

and solving (6) for $\underline{\tilde{x}}_i$ gives:

$$\tilde{x}_i \in c_i - \left(\underline{b}_i + \sum_{\substack{j=1 \\ j \neq i}}^n A_{ij} \cdot (x_j - c_j) \right) / A_{ii} \cap [z_i] \tag{7}$$

The Gauss-Seidel operator improves on (7), by observing that each iteration can use the previous interval in place of x_j , that is:

$$[y]^0 = [z]$$

$$[y_i]^{k+1} = \left(c_i - \left(\underline{b}_i + \sum_{\substack{j=1 \\ j \neq i}}^n A_{ij} \cdot (y_j)^k - c_j \right) \right) / A_{ii} \cap [y_i]^k \tag{8}$$

Equation (8) is the Gauss-Seidel operator ($y = N_{GS}([z])$).

A.2 EXISTENCE AND UNIQUENESS OF SOLUTIONS

Interval analysis techniques allow not only for the solution of systems of equations, but also provide theorems to check for the existence and the uniqueness of solutions within a given interval. The following theorem is proved in [16]: Let $F: D_0 \subseteq \mathfrak{R}^n \rightarrow \mathfrak{R}^n$ be continuously differentiable on $D \subseteq D_0$. If $\underline{x} \in [z] \in D$ and \underline{x}' denotes the Gauss-Seidel operator, then:

- (i) Every zero $\underline{x}^* \in [z]$ of F satisfies $\underline{x}^* \in \underline{x}'$.
- (ii) If $\underline{x}' \cap [z] = \emptyset$ then F contains no zero in x .
- (iii) If $\tilde{x} \in \text{int}([z])$ and $0 \neq \underline{x}' \subseteq \text{int}([z])$ then F contains a unique zero in x .

APPENDIX B - CONSTRAINT MODEL OF BEAM

There are nine dimensions which describe a cantilever beam:

- L = Length of beam (m), w = Width of beam top (m),
- t_1 = Height of beam top (m), t_2 = Height of beam bottom (m),
- t_3 = Width of beam bottom (m), σ_{\max} = Ultimate stress (Pa),
- n = Safety factor, E = Young's Modulus (Pa),
- ρ = Material density ($kg \cdot m^3$)

From those nine dimensions, numerous higher order attributes can be defined:

Volume: $L \cdot (t_2 t_3 + t_1 w)$

Mass: Volume * $\rho = \rho \cdot L \cdot (t_2 t_3 + t_1 w)$

Total Surface Area: $2L(t_1 + t_2 + w) + t_2 t_3 + t_1 w$

Maximum Moment:
$$M_{\max} = \frac{\sigma_{\max} I}{c} = \frac{\sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)}{12 \cdot \left(\frac{t_2^2 t_3}{2} + w t_1 \left(\frac{t_1}{2} + t_2 \right) \right)}$$

Maximum Load:
$$P_{\max} = \frac{2M_{\max}}{L^2} = \frac{\sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)}{3L^2 (t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w)}$$

Maximum Allowable Load:
$$P_{\text{allow}} = \frac{2M_{\max}}{nL^2} = \frac{\sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)}{3nL^2 (t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w)}$$

End Angle at Maximum Load:

$$\theta_{\max} = \frac{P_{\max} L^3}{6EI} = \frac{2L\sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)^2}{3E(t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w)(t_2^4 t_3^2 + 4t_1^3 t_2 t_3 w + 6t_1^2 t_2^2 t_3 w + 4t_1 t_2^3 t_3 w + t_1^4 w^2)}$$

End Angle at Maximum Allowable Load:

$$\theta_{\text{allow}} = \frac{P_{\text{allow}} L^3}{6EI} = \frac{2L\sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)^2}{3nE(t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w)(t_2^4 t_3^2 + 4t_1^3 t_2 t_3 w + 6t_1^2 t_2^2 t_3 w + 4t_1 t_2^3 t_3 w + t_1^4 w^2)}$$

End Deflection at Maximum Load:

$$\Delta_{\max} = \frac{P_{\max} L^4}{8EI} = \frac{L^2 \sigma_{\max} t_2^3 t_3 (t_2 t_3 + t_1 w)^2}{2E(t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w)(t_2^4 t_3^2 + 4t_1^3 t_2 t_3 w + 6t_1^2 t_2^2 t_3 w + 4t_1 t_2^3 t_3 w + t_1^4 w^2)}$$

End Deflection at Maximum Allowable Load:

$$\Delta_{allow} = \frac{P_{all} L^4}{8EI} = \frac{L^2 \sigma_{max} t_2^3 t_3 (t_2 t_3 + t_1 w)^2}{(2nE(t_2^2 t_3 + t_1^2 w + 2t_1 t_2 w) (t_2^4 t_3^2 + 4t_1^3 t_2 t_3 w + 6t_1^2 t_2^2 t_3 w + 4t_1 t_2^3 t_3 w + t_1^4 w^2))}$$

REFERENCES

1. Agrawal, R., A Constraint Management Approach for Optimal Design of Mechanical Systems, Ph.D. Thesis, Ohio State University, 1991.
2. Allgower and K. Georg, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, 1990.
3. Beaty, P, A. Fitzhorn, and G. J. Herron, "Extensions in Variational Geometry that Generate and Modify Object Edges Composed of Rational Bezier Curves", *Computer-Aided Design*, vol. 26, no. 2, February 1994, pp 98-108.
4. Borning, B., Freeman-Benson, and M. Wilson, "Constraint Hierarchies", *Lisp and Symbolic Computation*, 1992, pp 223-270.
5. Buchanan, A., and A. de Pennington, "Constraint Definition System: A Computer Algebra Based Approach to Solving Geometric Problems", *Computer Aided Design*, 1993, December, vol. 25, no. 12, pp. 741-750.
6. Gallaher, D. T., *Variational Systems in Computer-Aided Design*, M.S. Thesis, Massachusetts Institute of Technology, 1984.
7. Thompson, D. W., *On Growth and Form*, University Press, Second Edition, 1952.
8. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical Toolbox for Verified Computing I*, Springer-Verlag, Germany, 1993.
9. Kearfott, B. and Z. Xing, "An Interval Step Control for Continuation Methods", *SIAM Journal of Numerical Analysis*, June 1994, vol., 31, no. 3, pp. 892-914.
10. Jablokow, J.J. Uicker Jr., and D.A. Turcic, "Topological and Geometric Consistency in Boundary Representations of Solid Models of Mechanical Components", *Journal of Mechanical Design*, vol. 115, December 1993, pp 762-769.
11. Krawczyk and A. Neumaier, "Interval Slopes for Rational Functions and Associated Centered Forms", *SIAM Journal of Numerical Analysis*, June 1985, vol. 22, no. 3, pp. 604-616.
12. Light, R. A., *Symbolic Dimensioning in Computer-Aided Design*, M.S. Thesis, Massachusetts Institute of Technology, 1980.
13. Lin, V. C. *Three-Dimensional Variational Geometry in Computer-Aided Design*, M.S. Thesis, Massachusetts Institute of Technology, 1981.
14. Mackrell, J., "Making Sense of a Revolution", *Computer Graphics World*, November 1993, pp 26-38.
15. Neumaier, A., "Rigorous Sensitivity Analysis for Parameter-Dependent Systems of Equations", *Journal of Mathematical Analysis and Applications*, vol. 144 (1989), pp 16-25.
16. Neumaier, A., *Interval Methods for Systems of Equations*, Cambridge University Press, New York, NY, 1990.
17. Serrano, D., *MathPAK: An Interactive Preliminary Design Package*, M.S. Thesis, Massachusetts Institute of Technology, 1984.
18. Wampler, A. P. Morgan, and A. J. Sommese, "Numerical Continuation Methods for Solving Polynomial Systems Arising in Kinematics", *Journal of Mechanical Design*, March 1990, vol. 112, pp. 59-68.
19. Wilson, M. A. *Hierarchical Constraint Logic Programming*, Ph.D. Thesis, University of Washington, May 1993.
20. Suzuki, H., H. Ando, and F. Kimura, "Geometric Constraints and Reasoning for Geometrical CAD Systems", *Computers and Graphics*, vol. 14, no. 2, pp 211-224.
21. Huffaker, A. V. and O. Z. Maimon, "Maintaining a Consistent Configuration in a Constraint-Based Mechanical Design System," *ASME Annual Winter Meeting, RSAFP Symposium*, 1995.

CHAPTER 15

CONSTRAINT-BASED DESIGN OF FAIRED PARAMETRIC CURVES

This chapter demonstrates techniques to allow relations on parametric curves in a variational design system. Constraints on the curves, which are normally represented as constrained nonlinear optimization problems, are reduced to systems of nonlinear equations (using the necessary conditions of the Non-Linear Programming). Additional degrees of freedom are constrained through fairing the curve and the resulting NLP is also reduced to its necessary conditions. Although the solution set of the necessary conditions contains the optimum, it contains many other solutions as well. The COAST design consistency algorithm introduced in Chapter 14 is extended to handle consistency when constraints take the form of relations between objects. Examples are given for elementary curves and for an apparel design system.

15.1 CONSTRAINT-BASED CURVE DESIGN

Computer aided design is typically performed in a hierarchical manner, with more complex, *composite* objects being iteratively defined in terms of increasingly simpler objects. At the lowest level are *primitive* objects which usually consist of points, lines, arcs, and any other simple geometrical items. In constraint-based design, arrangements of primitive objects can be constrained using distance, angle, or any other mathematical relations to form a composite object. For these primitive objects, relations such as distance are computed through closed-form arithmetic expressions. Arrangements of composite objects can also be defined (e.g., assemblies) but constraints on composite objects are typically limited to relations between specific primitive components of the composite objects.

Despite the apparent distinction between primitive and composite objects, curves and other parametric objects have always held a rather ambiguous status in design systems. Mathematically, curves should be considered as primitive objects. They are fully defined through mathematical functions and are not composed of any other primitive objects. Additionally, relational constraints such as distance have obvious qualitative interpretations when applied to curves. However, there are no closed form expressions for simple relations so, instead, they are treated like composite objects. Accordingly, in constraint-based design systems, constraints describing the actual parametric objects are not allowed. Instead, curves can only be created by

individually creating each of the curve's defining points.

In this chapter, we demonstrate constraint-based design of parametric curves when the constraints are on the whole curve (e.g., a curve having a certain arc length or being a given distance from another graphical object) and not solely on the defining points of the curve. We introduce techniques that allow designers to define relations on parametric curves in a variational design system. Since design is an evolutionary process (see Chapter 2), the constraints will almost certainly be changed over time. These constraints on the curves, which are normally represented as constrained nonlinear optimization problems, are reduced to systems of nonlinear equations (using the necessary conditions of the NLP). Although the solution set of the necessary conditions contains the optimum, it contains many other solutions as well. The COAST design consistency algorithm developed in Chapter 14 is then implemented to maintain consistency. Finally, in order to demonstrate these techniques, an example is shown incorporating Bezier curves and an apparel design system is presented that can grade a design to any size.

15.2 PREVIOUS WORK

Variational design involves the simultaneous satisfaction of a system of constraints. In numerical approaches to constraint satisfaction, each constraint is expressed as an algebraic equation and the system of equations is solved through a numerical method, e.g. Newton-Raphson [1-3]. Curve creation in variational design systems typically involve individually constraining the defining points of the curve - constraints on the curve itself are not allowed.

Recent research in constraint-based curve design has focused on two aspects of curve creation: more powerful constraints [4-7] and better interactive techniques [8-10]. Nowacki and Lu [4] describe a technique for incorporating constraints on composite curves and demonstrate the ability to constrain the area enclosed by a composite curve. The constraints are approximated with quadrature techniques and the optimization problem resulting from fairing the curve is reduced to a system of nonlinear equations which is then solved. There are typically several possible curves, however, that would solve the constraints and there is no method presented for selecting a consistent one. We generally follow the techniques presented in this paper while also incorporating design consistency methods to allow the user to interact with the constraints while maintaining a consistent solution. Roulier [5] describes an algorithm to create Bezier curves of a given arc length. Although the given constraint scheme would typically result in many possible solutions, the algorithm assumes many aspects of the curve which reduce the solutions to a one-parameter family of curves at the expense of restricting the flexibility of the designer.

Research in interactive curve design seeks to create techniques for intelligent user control of the shape of the curve. The designer can constrain aspects of the curve (e.g., distance or tangency) anywhere along the curve. All of these systems, however, require the user to apply the constraint to a specific point on the curve (at a user-defined value of the curve parameter) rather than to the curve as a whole. The

research presented here, however, allows the user to define a constraint to be applied to the whole curve rather than to just a point on the curve.

15.3 MAINTAINING DESIGN CONSISTENCY IN CONSTRAINT-BASED CURVE DESIGN

In this section, methodology is demonstrated that allows designers to constrain the behavior of a curve as a whole. Using these techniques, a designer is able to constrain both intrinsic properties of a curve (e.g., arc length or total curvature) as well as extrinsic properties of the curve (e.g., distance from another object). Design consistency techniques are incorporated so the designer can then modify the constraints and obtain a new curve that is consistent with the original curve.

Unbounded graphical objects are represented as a vector of dimensions, \underline{G} . For the purpose of this chapter, points, lines, and circles are defined with the following dimensions (other dimensions are certainly possible):

- Point: $\underline{G}=(x,y)$ [Cartesian coordinates]
- Line: $\underline{G}=(x,y,ang)$ [Passing through a point with a given angle]
- Circle: $\underline{G}=(x,y,r)$ [Centered about a point with a given radius]

Bounded graphical objects (e.g. line, arc, or curve segment) are represented by an unbounded graphical object and an additional parameter that describes the endpoints of the object:

- Line Segment: $\underline{D}=(\underline{G},s)=(x,y,ang,s)$
- Arc: $\underline{D}=(\underline{G},s)=(x,y,r,s)$
- 2-D Cubic Bezier Curve:

$$\underline{D}=(\underline{G},s)=\left(P_{0x}, P_{1x}, P_{2x}, P_{3x}, P_{0y}, P_{1y}, P_{2y}, P_{3y}, s\right)$$

Any constraint on a graphical object is transformed into an algebraic expression in terms of the object's dimensions,

$$\theta_i = f(\underline{D}) = \begin{cases} f(\underline{G}) & \text{unbounded} \\ f(\underline{G},s) & \text{bounded} \end{cases}$$

Satisfying a constraint on a graphical object requires calculating the dimension values which satisfy the expressions. Relations, such as distance, between unbounded objects (e.g., point, line, and circle) can be computed through closed-form arithmetic expressions. Closed form expressions are not possible, however, when the relations involving the majority of bounded objects. Accordingly, in most constraint-based design systems, constraints describing whole curves are not allowed. Instead, curves can only be created by individually creating each of the defining points.

15.3.1 DISTANCE CONSTRAINTS

Consider the problem of constraining the distance between a parametric curve, $\underline{P} = (\underline{G}, s)$, and a point, $\underline{q} = (x, y)$ (see Figure 15.1a). The constraint is defined by calculating the distance between a point on the curve, \underline{P} , and the point, \underline{q} :

$$\theta_1 = \sqrt{(P_x(\underline{G}, s) - q_x)^2 + (P_y(\underline{G}, s) - q_y)^2} = d.$$

Because the user is not required to define the specific value of s , the distance is found by optimizing the objective function, $\sqrt{(P_x(\underline{G}, s) - q_x)^2 + (P_y(\underline{G}, s) - q_y)^2}$ as s varies between its bounds (0 to 1 for normalized curves): $d = \underset{s}{\text{opt}} f(\underline{G}, s, \underline{q})$. If the

distance between the point and the circle is defined as the global minimum of the distance function between the two objects, then the curve can only be constrained in one location as shown in Figure 15.1a. There are other locations on the curve, however, where the designer may wish to constrain the distance (Figure 15.1b). These locations can be categorized as locations where the gradient of the distance function with respect to the curve parameter is zero. Graphically, the locations are those points on the curve where the normal of the curve intersects the point. The required algebraic constraint for these locations is:

$$\theta_2 = \frac{\partial \sqrt{(P_x(\underline{G}, s) - q_x)^2 + (P_y(\underline{G}, s) - q_y)^2}}{\partial s} = 0$$

The second constraint serves to restrict the locations at which the distance can be measured. These locations are seen to be the local optima (both minima and maxima) of the initial optimization problem.

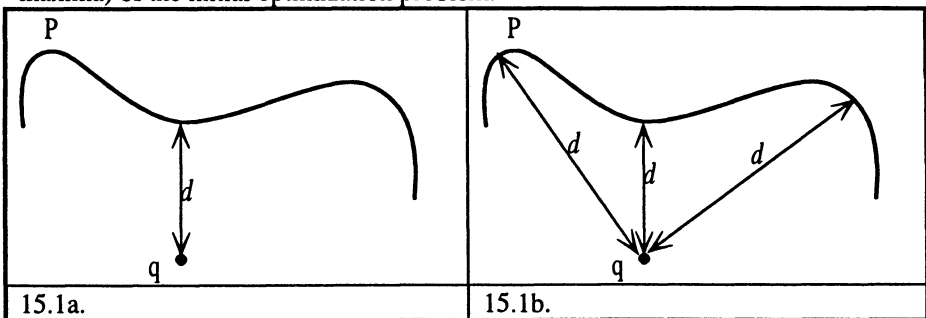


Figure 15.1 Distance Between a Point and a Curve

In general, the local optima is found by analyzing the *necessary conditions* of the optimization problem. For an unconstrained optimization problem,

$$\text{opt } f(\underline{G}, s)$$

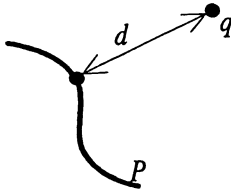
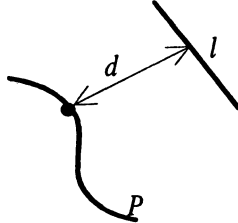
(where **opt** is either minimize or maximize and $f(\underline{G}, s)$ is an algebraic expression), the necessary conditions are:

$$\nabla f(\underline{G}, s) = 0,$$

which, in this example, yields the second constraint.

Notice that the bounds on the curve parameter are not present in these necessary conditions. The first time the constraints are solved, the designer can help the program find the desired optimum. After subsequent modifications, design consistency techniques are used to maintain the desired local optimum and the value of the parameter is simply checked against the bounds to ensure its validity.

Figure 15.2 shows the objective functions for constraining the distance between a parametric curve, P , and a point, line, circle, and another parametric curve. The local optima are calculated by finding the points along the curve(s) at which the gradients of the objective function with respect to the curve parameter(s).

	
$d = \text{opt}_s \left(f(\underline{G}, s, \underline{q}) \right) =$ $\text{opt}_s \left(\sqrt{(P_x(\underline{G}, s) - q_x)^2 + (P_y(\underline{G}, s) - q_y)^2} \right)$	$d = \text{opt}_s \left(f(\underline{G}, s, l) \right) =$ $\text{opt}_s \left(\sqrt{\left((l_y - P_x(\underline{G}, s)) \cos(l_{ang}) - (l_x - P_y(\underline{G}, s)) \sin(l_{ang}) \right)^2} \right)$
<p>2a.</p>	<p>2b.</p>

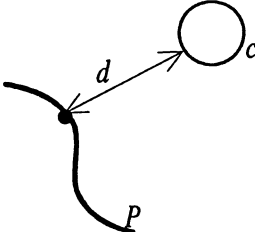
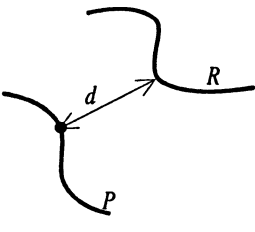
	
$d = \underset{s}{opt} (f(\underline{G}, s, c)) =$ $\underset{s}{opt} \left(\sqrt{\left(P_x(\underline{G}, s) - c_x \right)^2 + \left(P_y(\underline{G}, s) - c_y \right)^2} - c_r \right)$	$d = \underset{s,t}{opt} (f(\underline{G}_1, s, \underline{G}_2, t)) =$ $\underset{s,t}{opt} \left(\sqrt{\left(P_x(\underline{G}_1, s) - R_x(\underline{G}_2, t) \right)^2 + \left(P_y(\underline{G}_1, s) - R_y(\underline{G}_2, t) \right)^2} \right)$
<p>2c.</p>	<p>2d.</p>

Figure 15.2 Objective Functions for Calculating Distance using Optimization

15.3.2 ARC LENGTH

Intrinsic integral properties of a curve are typically calculated by integrating over the length of the curve. For example, the arc length of a parametric 2-D curve can be

calculated as $\theta_1 = f(\underline{G}, s) = \int_{s=0}^1 \sqrt{\left[\frac{\partial(P_x(\underline{G}, s))}{\partial s} \right]^2 + \left[\frac{\partial(P_y(\underline{G}, s))}{\partial s} \right]^2} ds$ which can be

approximated with a nonlinear equation using an extended quadrature technique [11].

15.3.3 CONSISTENCY IN CURVE FAIRING

Consider a cubic Bezier curve with endpoints at (0,4) and (3,0) that is constrained to be 0.25 away from circle c (Figure 15.3). The curve has eight unknowns, five of which are constrained, leaving three degrees of freedom to optimize a performance index (here, the arc length of the curve is minimized). This situation gives the curve shown in Figure 15.3a. Now move the circle. In order to maintain the distance constraint, the curve is modified (Figure 15.3b). At some point, after moving the circle a sufficient amount, the curve will jump to the global optimum on the opposite side of the circle (Figure 15.3c). In an interactive design system, however, this is typically not the desired behavior. Instead, solution consistency should be maintained. In this case, the local optimum that maintains design consistency is on

the same side of the circle (Figure 15.3d).

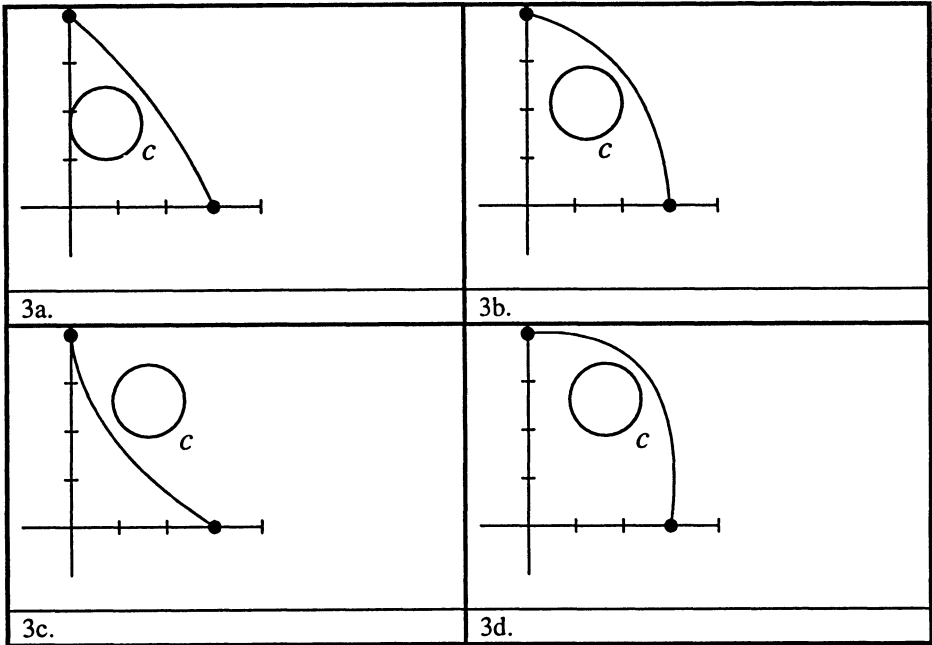


Figure 15.3 Global vs. Local Optimum

We assert that after a constraint modification in an interactive design system, the new intended (consistent) solution is often a locally optimized solution, rather than a globally optimized one. In an interactive design system, the designer is constantly modifying the constraints in search of a good design. After each modification, the designer expects the design to be changed in a predictable manner. Small changes in the constraints should lead to small changes in the design. To locate the local optima, we again look to the necessary conditions of the NLP. For a constrained optimization problem,

$$\begin{aligned} &\text{opt } obj(\underline{G}, s) \\ &\text{such that } \underline{h}(\underline{G}, s) = 0, \end{aligned}$$

the necessary conditions are:

$$\begin{aligned} \nabla obj(\underline{G}, s) + \lambda^T \nabla \underline{h}(\underline{G}, s) &= 0 \\ \underline{h}(\underline{G}, s) &= 0 \end{aligned}$$

The COAST method is then utilized to maintain a consistent solution to the system of

nonlinear equations.

15.3.4 COAST METHODOLOGY FOR DESIGN CONSISTENCY

At each design iteration, we begin with a design that both satisfies the constraints and is the intended solution. When the constraints are modified, we wish to not only find a new satisfactory design, we wish to find the new *intended* design. This is what we mean by *consistent design* (see chapter 14). If there is only one possible solution to the constraints, then it is easy to maintain a consistent design. It is much harder if there are multiple competing solutions, all of which satisfy the constraints.

Fortunately, this iterative view of design directs us towards a principle of design consistency: *small changes in specifications should lead to small changes in design*. Design is not a chaotic environment (where small changes in input lead to large changes in output). Furthermore, large changes in specifications can often be decomposed to a series of small changes, in which case the principle can still be applied.

A system of constraints is created from the user-defined constraints $(\theta | \theta_i = f_i(\underline{G}_i, s_i))$. When the constraints are modified, let \underline{t} be the vector of modified constraint coefficients

$$\left(\theta(\underline{D}, \underline{t}_0) \rightarrow \theta(\underline{D}, \underline{t}_1), \underline{D} | \underline{D}_i = \begin{cases} (\underline{G}_i, s_i) & \text{for distance and other relations between curves} \\ (\underline{G}_i) & \text{for arc length and other integral properties} \end{cases} \right)$$

By reducing the optimization problem to a system of nonlinear equations, we have simplified the problem at the expense of introducing many more possible solutions. By rigorously analyzing the sensitivity of the system of equations as in Chapter 14, we can follow the solution trajectory as the constraints change. The COAST methodology attempts to maintain a consistent solution through the analysis of the solution trajectory of the desired solution. When constraints are modified, a homotopy is setup between the two systems of equations, $\theta(\underline{D}, \underline{t}_0) = 0$ and $\theta(\underline{D}, \underline{t}_1) = 0$ as $\theta(\underline{D}, \underline{t}_0 + \alpha(\underline{t}_1 - \underline{t}_0)) = 0$ and α is varied from 0 to 1.

We assume that the coefficients of the constraints can be changed, but not the form of the constraints. The rigorous sensitivity of systems of equations of the form $\theta(\underline{D}, \underline{t}) = 0$ is described in [12]. Using those methods, we extend the COAST methodology developed in Chapter 14 to handle modifications to any coefficients of the constraints (Figure 15.4).

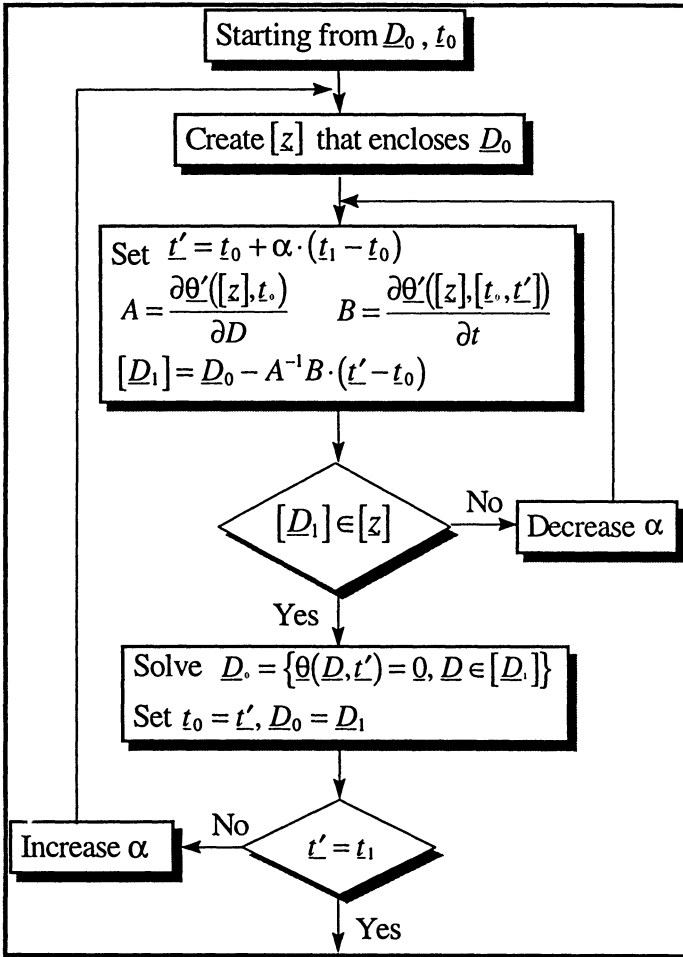


Figure 15.4 COAST Algorithm for Implicit Constraint Changes

15.4 EXAMPLES

15.4.1 BEZIER CURVE FROM DISTANCE CONSTRAINTS

Problem: Create a 2-D Bezier curve with endpoints at (-1,2) and (5,-1) such that it passes 0.25 from the point, $q_1 = (1,1)$, and 0.5 from the point, $q_2 = (3,4)$ (Figure 15.5). Fair the curve by minimizing the arc length. Then, move q_1 to (1,4). The curve should be adjusted in a consistent manner.

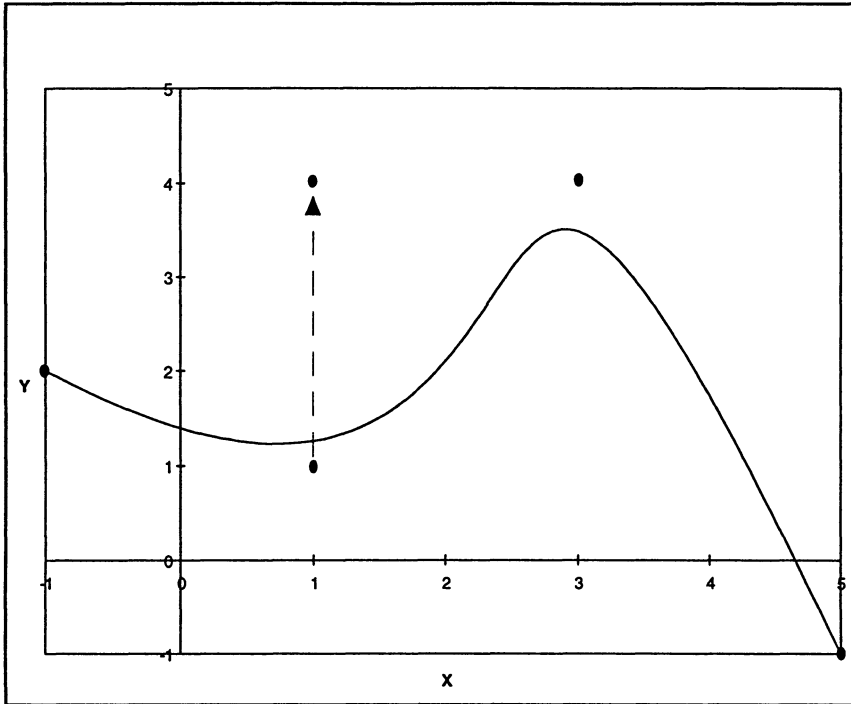


Figure 15.5 Bezier Curve from Distance Constraints

The parametric function for cubic Bezier curves is:

$$P(\underline{G}, s) = (1-s)^3 P_0 + 3(1-s)^2 s P_1 + 3(1-s) s^2 P_2 + s^3 P_3 \quad \text{where}$$

$$\underline{G} = [P_{0x}, P_{1x}, P_{2x}, P_{3x}, P_{0y}, P_{1y}, P_{2y}, P_{3y}], \quad P(\underline{G}, s) = \begin{bmatrix} P_x(\underline{G}, s) \\ P_y(\underline{G}, s) \end{bmatrix} \quad (\text{in 2-D}), \quad \text{and}$$

$0 \leq s \leq 1$. The endpoint constraints yield four simple equations:

$$\theta_1: P_x(\underline{G}, 0) = P_{0x} = -1$$

$$\theta_2: P_y(\underline{G}, 0) = P_{0y} = 2$$

$$\theta_3: P_x(\underline{G}, 1) = P_{3x} = 5$$

$$\theta_4: P_y(\underline{G}, 1) = P_{3y} = -1$$

Each distance constraint adds two equations and one unknown:

$$\theta_5: \sqrt{(P_x(\underline{G}, s_1) - q_{1x})^2 + (P_y(\underline{G}, s_1) - q_{1y})^2} = 0.25$$

$$\theta_6: \frac{\partial \sqrt{\left(P_x(\underline{G}, s_1) - q_{1x}\right)^2 + \left(P_y(\underline{G}, s_1) - q_{1y}\right)^2}}{\partial s_1} = 0$$

$$\theta_7: \sqrt{\left(P_x(\underline{G}, s_2) - q_{2x}\right)^2 + \left(P_y(\underline{G}, s_2) - q_{2y}\right)^2} = 0.5$$

$$\theta_8: \frac{\partial \sqrt{\left(P_x(\underline{G}, s_2) - q_{2x}\right)^2 + \left(P_y(\underline{G}, s_2) - q_{2y}\right)^2}}{\partial s_2} = 0$$

for a total of eight equations and ten unknowns. The remaining unknowns are constrained by minimizing the arc length of the curve. We approximate the arc length, L , of the curve with a 16 point trapezoid quadrature formula [11] that estimates the integral given previously and use that equation to form the objective function for the NLP, giving the following system of necessary conditions:

$$\nabla \text{Obj}(\underline{G}, s) + \lambda^T \nabla h(\underline{G}, s) = 0 \Rightarrow$$

$$\theta_9: \frac{\partial L}{\partial P_{0x}} + \lambda_1 \frac{\partial \theta_1}{\partial P_{0x}} + \lambda_2 \frac{\partial \theta_2}{\partial P_{0x}} + \lambda_3 \frac{\partial \theta_3}{\partial P_{0x}} + \lambda_4 \frac{\partial \theta_4}{\partial P_{0x}} + \lambda_5 \frac{\partial \theta_5}{\partial P_{0x}} + \lambda_6 \frac{\partial \theta_6}{\partial P_{0x}} + \lambda_7 \frac{\partial \theta_7}{\partial P_{0x}} + \lambda_8 \frac{\partial \theta_8}{\partial P_{0x}} = 0$$

⋮

$$\theta_{12}: \frac{\partial L}{\partial P_{3x}} + \lambda_1 \frac{\partial \theta_1}{\partial P_{3x}} + \lambda_2 \frac{\partial \theta_2}{\partial P_{3x}} + \lambda_3 \frac{\partial \theta_3}{\partial P_{3x}} + \lambda_4 \frac{\partial \theta_4}{\partial P_{3x}} + \lambda_5 \frac{\partial \theta_5}{\partial P_{3x}} + \lambda_6 \frac{\partial \theta_6}{\partial P_{3x}} + \lambda_7 \frac{\partial \theta_7}{\partial P_{3x}} + \lambda_8 \frac{\partial \theta_8}{\partial P_{3x}} = 0$$

$$\theta_{13}: \frac{\partial L}{\partial P_{0y}} + \lambda_1 \frac{\partial \theta_1}{\partial P_{0y}} + \lambda_2 \frac{\partial \theta_2}{\partial P_{0y}} + \lambda_3 \frac{\partial \theta_3}{\partial P_{0y}} + \lambda_4 \frac{\partial \theta_4}{\partial P_{0y}} + \lambda_5 \frac{\partial \theta_5}{\partial P_{0y}} + \lambda_6 \frac{\partial \theta_6}{\partial P_{0y}} + \lambda_7 \frac{\partial \theta_7}{\partial P_{0y}} + \lambda_8 \frac{\partial \theta_8}{\partial P_{0y}} = 0$$

⋮

$$\theta_{16}: \frac{\partial L}{\partial P_{3y}} + \lambda_1 \frac{\partial \theta_1}{\partial P_{3y}} + \lambda_2 \frac{\partial \theta_2}{\partial P_{3y}} + \lambda_3 \frac{\partial \theta_3}{\partial P_{3y}} + \lambda_4 \frac{\partial \theta_4}{\partial P_{3y}} + \lambda_5 \frac{\partial \theta_5}{\partial P_{3y}} + \lambda_6 \frac{\partial \theta_6}{\partial P_{3y}} + \lambda_7 \frac{\partial \theta_7}{\partial P_{3y}} + \lambda_8 \frac{\partial \theta_8}{\partial P_{3y}} = 0$$

$$\theta_{17}: \frac{\partial L}{\partial s_1} + \lambda_1 \frac{\partial \theta_1}{\partial s_1} + \lambda_2 \frac{\partial \theta_2}{\partial s_1} + \lambda_3 \frac{\partial \theta_3}{\partial s_1} + \lambda_4 \frac{\partial \theta_4}{\partial s_1} + \lambda_5 \frac{\partial \theta_5}{\partial s_1} + \lambda_6 \frac{\partial \theta_6}{\partial s_1} + \lambda_7 \frac{\partial \theta_7}{\partial s_1} + \lambda_8 \frac{\partial \theta_8}{\partial s_1} = 0$$

$$\theta_{18}: \frac{\partial L}{\partial s_2} + \lambda_1 \frac{\partial \theta_1}{\partial s_2} + \lambda_2 \frac{\partial \theta_2}{\partial s_2} + \lambda_3 \frac{\partial \theta_3}{\partial s_2} + \lambda_4 \frac{\partial \theta_4}{\partial s_2} + \lambda_5 \frac{\partial \theta_5}{\partial s_2} + \lambda_6 \frac{\partial \theta_6}{\partial s_2} + \lambda_7 \frac{\partial \theta_7}{\partial s_2} + \lambda_8 \frac{\partial \theta_8}{\partial s_2} = 0$$

which, along with the eight original constraints, forms a system of 18 equations and 18 unknowns. As we move along the homotopy, the curve is adjusted accordingly as shown in Figure 15.6. The final solution (shown in Figure 15.7) maintains a consistent design.

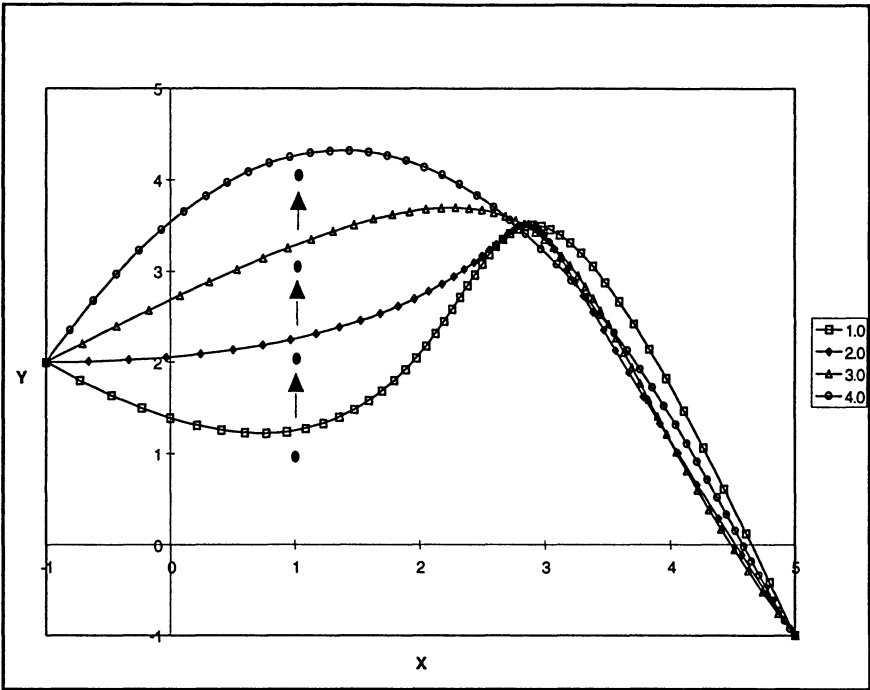


Figure 15.6 Sample Curves along the Homotopy

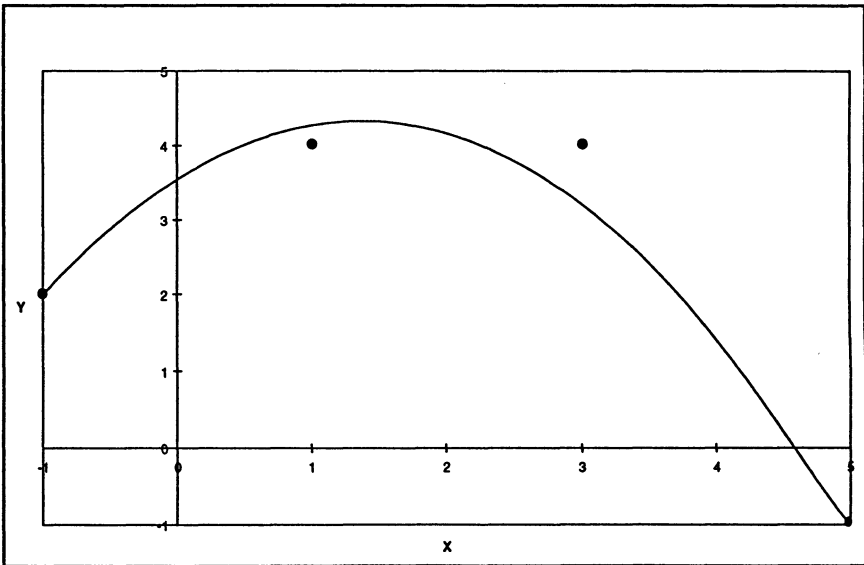


Figure 15.7 Ending Curve

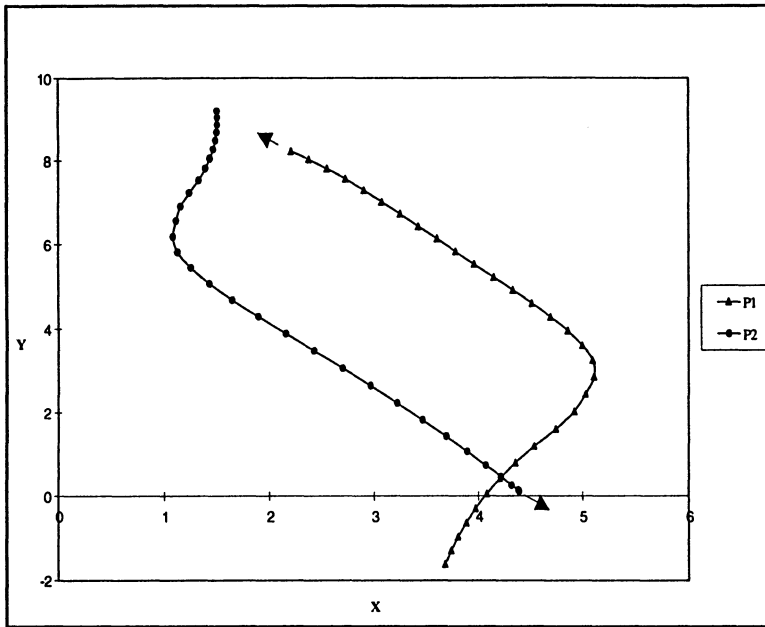


Figure 15.8 Solution Trajectories of Defining Points

Finally, the solution trajectories of the center two control points along the homotopy are shown in Figure 15.8.

15.4.2 APPAREL DESIGN

Apparel design is the modification of two-dimensional patterns so that when they are assembled, form unique or attractive styles. Apparel design begins with the definition of a basic pattern which is derived from the body measurements (the sloper). The definition of the sloper is a step-by-step process in which points, lines, circles, and curves are defined relative to the body measurements and existing elements. The end result is a collection of polygons or outlines which form a simple pattern. The patternmaker then performs a sequence of standard patternmaking operations on the outlines to modify the style of the piece.

While there are many apparel design CAD packages available, they all require the designer to manually define how each point of the design should be adjusted for different sizes. Using our techniques, it is possible to define a design in terms of the body measurements. Then, when the body measurements are modified, the design is updated in a consistent manner. For example, the following are a few steps of the design of a basic bodice sloper using the measurements, Bust Span, Bust Arc, and Side Length [13]:

O-P = Bust Span. Square from center front through N. (P is bust point).

- O-Q = Bust Arc. To find Q, measure from O to P to Q, with Q touching somewhere on J guideline indicated by broken line
 P-R = P-Q, plus 1/4 inch. R touches somewhere on I guideline indicated by broken line.
 R-S = Side Length. (R to S through Q)
 R-T = 1/2 inch.

Figure 15.9 shows most of the construction lines used in the creation of the front bodice pattern.

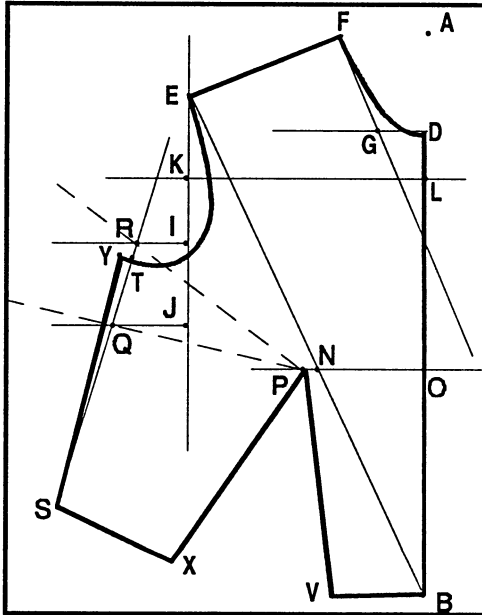


Figure 15.9 Bodice Sloper Construction

Unfortunately, however, the sequence does not always create an accurate design. For example, the angles between the points P and S and between P and B should be supplementary so that they are smooth when the design is folded over the body. The sequence given, however, does not always maintain this relationship. In order to maintain this relationship, we incorporate variational constraints on the angle of the segments. Additionally, we create the arm hole and neck hole curves using distance constraints from points M and G, respectively. Creating the front bodice requires 12 measurements (shown in Table 15.1 with their values, in inches, for sizes 10 and 12 as measured on sample dressforms):

Table 15.1 Bodice Measurements

Full Length 10: 17.3 12: 17.3	Across Shoulder 10: 7.3 12: 7.8	Center Front Length 10: 14.3 12: 14.8	Shoulder Slope 10: 17.0 12: 17.5
Shoulder Length 10: 5.1 12: 5.4	Arm Hole Depth 10: 4.8 12: 4.8	Bust Depth 10: 9.3 12: 9.4	Bust Span 10: 3.6 12: 4.1
Bust Arc 10: 9.3 12: 9.9	Side Length 10: 8.4 12: 9.2	Dart Placement 10: 2.8 12: 3.3	Waist Arc 10: 6.8 12: 7.3

The resulting system contains 54 equations (shown in Table 15.2):

Table 15.2 Bodice Constraints

$\theta_1: a_x=0$	$\theta_{28}: o_y = n_y$
$\theta_2: a_y=0$	$\theta_{29}: p_y = o_y$
$\theta_3: b_x=0$	$\theta_{30}: \text{dist}[o_x, o_y, p_x, p_y] = \text{Bust Span}$
$\theta_4: \text{dist}[a_x, a_y, b_x, b_y] = \text{Full Length}$	$\theta_{31}: q_y = j_y$
$\theta_5: c_y=0$	$\theta_{32}: \text{dist}[q_x, q_y, p_x, p_y] + \text{dist}[p_x, p_y, o_x, o_y] = \text{Bust Arc}$
$\theta_6: \text{dist}[a_x, a_y, c_x, c_y] = \text{Across Shoulder}$	$\theta_{33}: r_y = i_y$
$\theta_7: d_x=0$	$\theta_{34}: \text{dist}[r_x, r_y, p_x, p_y] = \text{dist}[q_x, q_y, p_x, p_y] + 1/4$
$\theta_8: \text{dist}[b_x, b_y, d_x, d_y] = \text{Center Front Length}$	$\theta_{35}: \text{dist}[r_x, r_y, s_x, s_y] = \text{Side Length}$
$\theta_9: e_x = c_x$	$\theta_{36}: q, r, \text{ and } s \text{ collinear}$
$\theta_{10}: \text{dist}[b_x, b_y, e_x, e_y] = \text{Shoulder Slope}$	$\theta_{37}: \text{dist}[r_x, r_y, t_x, t_y] = 1/2$
$\theta_{11}: f_y=0$	$\theta_{38}: r, s, \text{ and } t \text{ collinear}$
$\theta_{12}: \text{dist}[e_x, e_y, f_x, f_y] = \text{Shoulder Length}$	$\theta_{39}: \text{dist}[b_x, b_y, v_x, v_y] = \text{Dart Placement}$
$\theta_{13}: g_y=d_y$	$\theta_{40}: p, v, \text{ and } b \text{ orthogonal}$
$\theta_{14}: e, f, \text{ and } g \text{ orthogonal}$	$\theta_{41}: \text{dist}[s_x, s_y, x_x, x_y] = \text{Waist Arc} - \text{Dart Placement}$
$\theta_{15}: i_x = e_x$	$\theta_{42}: s, x, \text{ and } p \text{ orthogonal}$
$\theta_{16}: \text{dist}[e_x, e_y, i_x, i_y] = \text{Arm Hole Depth}$	$\theta_{43}: \text{dist}[y_x, y_y, t_x, t_y] = 1/2$
$\theta_{17}: j_x=i_x$	$\theta_{44}: s, t, \text{ and } y \text{ orthogonal}$
$\theta_{18}: j_y=i_y-2.$	$\theta_{45}: P_x(0) = e_x$
$\theta_{19}: k_x = e_x$	$\theta_{46}: P_x(1) = y_x$
$\theta_{20}: k_y = (e_y+i_y)/2.$	$\theta_{47}: P_y(0) = e_y$
$\theta_{21}: l_x = a_x$	$\theta_{48}: P_y(1) = y_y$

θ_{22} : $l_y = k_y$	θ_{49} : $\text{dist}[P_x(s_1), P_y(s_1), k_x, k_y] = 1/2$
θ_{23} : $m_x = k_x + 1/2$	θ_{50} : $R_x(0) = f_x$
θ_{24} : $m_y = k_y$	θ_{51} : $R_x(1) = d_x$
θ_{25} : $\text{dist}[e_x, e_y, n_x, n_y] = \text{Bust Depth}$	θ_{52} : $R_y(0) = f_y$
θ_{26} : $e, n, \text{ and } b \text{ collinear}$	θ_{53} : $R_y(1) = d_y$
θ_{27} : $o_x = a_x$	θ_{54} : $\text{dist}[R_x(s_2), R_y(s_2), g_x, g_y] = 1/2$

where $P(s)$ is the arm hole curve and $R(t)$ is the neck hole curve. From the constraint on the arm hole curve, θ_{49} , the necessary condition for the distance relation is added:

$$\theta_{55}: \frac{\partial \text{dist}[P_x(\underline{G}, s_1), P_y(\underline{G}, s_1), \underline{k}]}{\partial s_1} = 0.$$

To fair the curve, the stiffness criterion ($J = \frac{1}{2} \cdot \int (\|P''(\underline{G}, s)\|^2 + \|P'''(\underline{G}, s)\|^2) ds$) is used and estimated using an eight-point Simpson's quadrature formula to obtain the following necessary conditions (creating a Bezier curve):

$$\theta_{56}: \frac{\partial J}{\partial P_{0x}} + \lambda_1 \frac{\partial \theta_{45}}{\partial P_{0x}} + \lambda_2 \frac{\partial \theta_{46}}{\partial P_{0x}} + \lambda_3 \frac{\partial \theta_{47}}{\partial P_{0x}} + \lambda_4 \frac{\partial \theta_{48}}{\partial P_{0x}} + \lambda_5 \frac{\partial \theta_{49}}{\partial P_{0x}} + \lambda_6 \frac{\partial \theta_{55}}{\partial P_{0x}} = 0$$

⋮

$$\theta_{59}: \frac{\partial J}{\partial P_{3x}} + \lambda_1 \frac{\partial \theta_{45}}{\partial P_{3x}} + \lambda_2 \frac{\partial \theta_{46}}{\partial P_{3x}} + \lambda_3 \frac{\partial \theta_{47}}{\partial P_{3x}} + \lambda_4 \frac{\partial \theta_{48}}{\partial P_{3x}} + \lambda_5 \frac{\partial \theta_{49}}{\partial P_{3x}} + \lambda_6 \frac{\partial \theta_{55}}{\partial P_{3x}} = 0$$

$$\theta_{60}: \frac{\partial J}{\partial P_{0y}} + \lambda_1 \frac{\partial \theta_{45}}{\partial P_{0y}} + \lambda_2 \frac{\partial \theta_{46}}{\partial P_{0y}} + \lambda_3 \frac{\partial \theta_{47}}{\partial P_{0y}} + \lambda_4 \frac{\partial \theta_{48}}{\partial P_{0y}} + \lambda_5 \frac{\partial \theta_{49}}{\partial P_{0y}} + \lambda_6 \frac{\partial \theta_{55}}{\partial P_{0y}} = 0$$

⋮

$$\theta_{63}: \frac{\partial J}{\partial P_{3y}} + \lambda_1 \frac{\partial \theta_{45}}{\partial P_{3y}} + \lambda_2 \frac{\partial \theta_{46}}{\partial P_{3y}} + \lambda_3 \frac{\partial \theta_{47}}{\partial P_{3y}} + \lambda_4 \frac{\partial \theta_{48}}{\partial P_{3y}} + \lambda_5 \frac{\partial \theta_{49}}{\partial P_{3y}} + \lambda_6 \frac{\partial \theta_{55}}{\partial P_{3y}} = 0$$

$$\theta_{64}: \frac{\partial J}{\partial s_1} + \lambda_1 \frac{\partial \theta_{45}}{\partial s_1} + \lambda_2 \frac{\partial \theta_{46}}{\partial s_1} + \lambda_3 \frac{\partial \theta_{47}}{\partial s_1} + \lambda_4 \frac{\partial \theta_{48}}{\partial s_1} + \lambda_5 \frac{\partial \theta_{49}}{\partial s_1} + \lambda_6 \frac{\partial \theta_{55}}{\partial s_1} = 0$$

and likewise for the constraint on the neck hole curve:

$$\theta_{65}: \frac{\partial \text{dist}[R_x(\underline{G}, s_2), R_y(\underline{G}, s_2), \underline{g}]}{\partial s_2} = 0$$

$$\theta_{66}: \frac{\partial J}{\partial P_{0x}} + \lambda_7 \frac{\partial \theta_{50}}{\partial P_{0x}} + \lambda_8 \frac{\partial \theta_{51}}{\partial P_{0x}} + \lambda_9 \frac{\partial \theta_{52}}{\partial P_{0x}} + \lambda_{10} \frac{\partial \theta_{53}}{\partial P_{0x}} + \lambda_{11} \frac{\partial \theta_{54}}{\partial P_{0x}} + \lambda_{12} \frac{\partial \theta_{65}}{\partial P_{0x}} = 0$$

⋮

$$\theta_{69}: \frac{\partial J}{\partial P_{3x}} + \lambda_7 \frac{\partial \theta_{50}}{\partial P_{3x}} + \lambda_8 \frac{\partial \theta_{51}}{\partial P_{3x}} + \lambda_9 \frac{\partial \theta_{52}}{\partial P_{3x}} + \lambda_{10} \frac{\partial \theta_{53}}{\partial P_{3x}} + \lambda_{11} \frac{\partial \theta_{54}}{\partial P_{3x}} + \lambda_{12} \frac{\partial \theta_{65}}{\partial P_{3x}} = 0$$

$$\theta_{70}: \frac{\partial J}{\partial P_{0y}} + \lambda_7 \frac{\partial \theta_{50}}{\partial P_{0y}} + \lambda_8 \frac{\partial \theta_{51}}{\partial P_{0y}} + \lambda_9 \frac{\partial \theta_{52}}{\partial P_{0y}} + \lambda_{10} \frac{\partial \theta_{53}}{\partial P_{0y}} + \lambda_{11} \frac{\partial \theta_{54}}{\partial P_{0y}} + \lambda_{12} \frac{\partial \theta_{65}}{\partial P_{0y}} = 0$$

⋮

$$\theta_{73}: \frac{\partial J}{\partial P_{3y}} + \lambda_7 \frac{\partial \theta_{50}}{\partial P_{3y}} + \lambda_8 \frac{\partial \theta_{51}}{\partial P_{3y}} + \lambda_9 \frac{\partial \theta_{52}}{\partial P_{3y}} + \lambda_{10} \frac{\partial \theta_{53}}{\partial P_{3y}} + \lambda_{11} \frac{\partial \theta_{54}}{\partial P_{3y}} + \lambda_{12} \frac{\partial \theta_{65}}{\partial P_{3y}} = 0$$

$$\theta_{74}: \frac{\partial J}{\partial s_2} + \lambda_7 \frac{\partial \theta_{50}}{\partial s_2} + \lambda_8 \frac{\partial \theta_{51}}{\partial s_2} + \lambda_9 \frac{\partial \theta_{52}}{\partial s_2} + \lambda_{10} \frac{\partial \theta_{53}}{\partial s_2} + \lambda_{11} \frac{\partial \theta_{54}}{\partial s_2} + \lambda_{12} \frac{\partial \theta_{65}}{\partial s_2} = 0$$

There are 22 points with two dimensions each (44 unknowns) and for each curve there are eight curve parameters (16 unknowns), 6 λ 's (12 unknowns), and one curve parameter (2 unknowns) for a total of 74 unknowns.

Entering the dimensions measured from a size 10 dressform and solving the system of equations (with an adequate starting point) yields the solution shown in Figure 15.10.

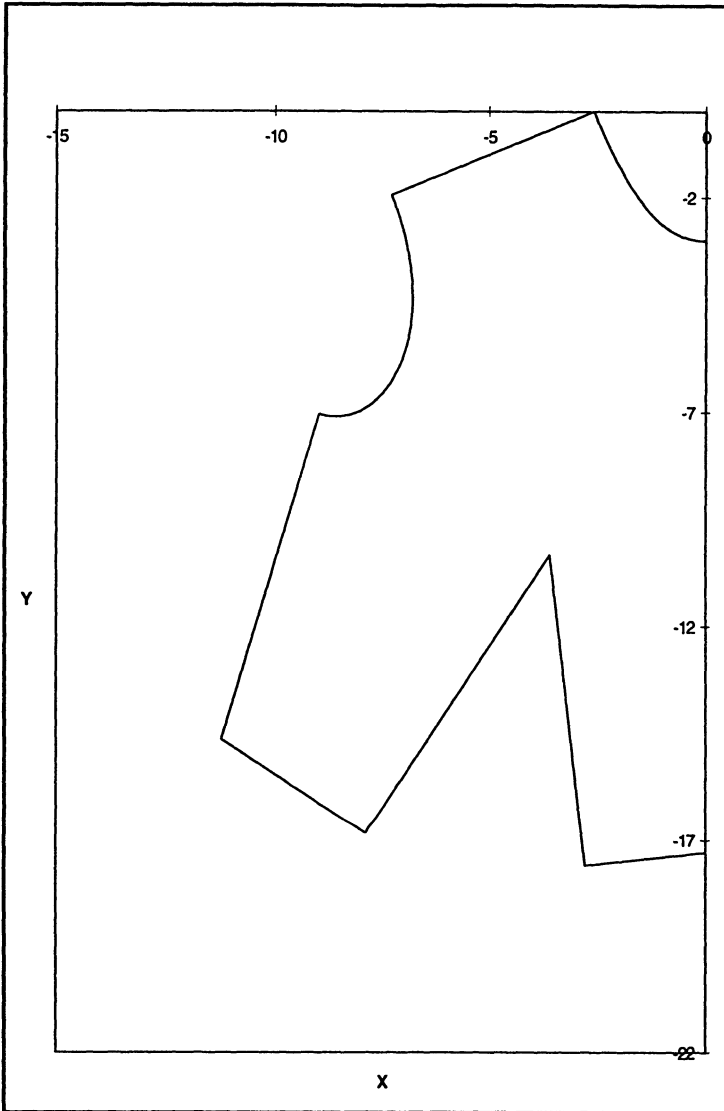


Figure 15.10 Bodice - Size 10

As we move the measurements from size 10 to size 12 along the homotopy, the COAST method is able to converge to a consistent solution (shown in Figure 15.11).

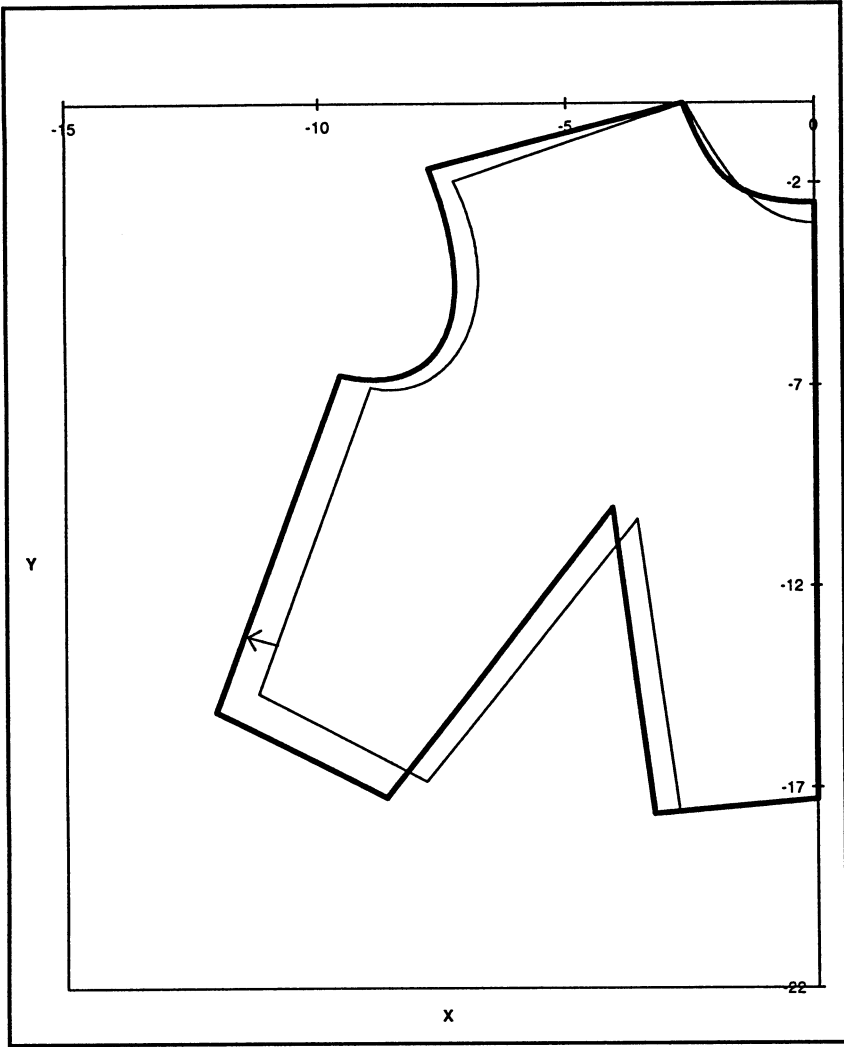


Figure 15.11 Bodice - Size 10 to 12

The trajectories of the center two defining points of the arm hole and neck hole curves are shown in Figures 15.12 and 15.13.

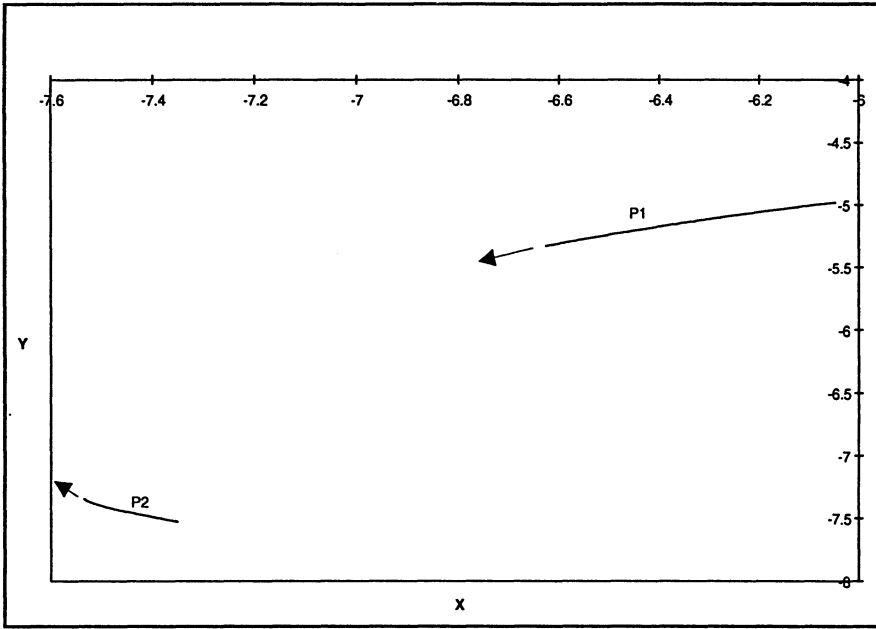


Figure 15.12 Solution Trajectory of Arm Hole Curve Defining Points

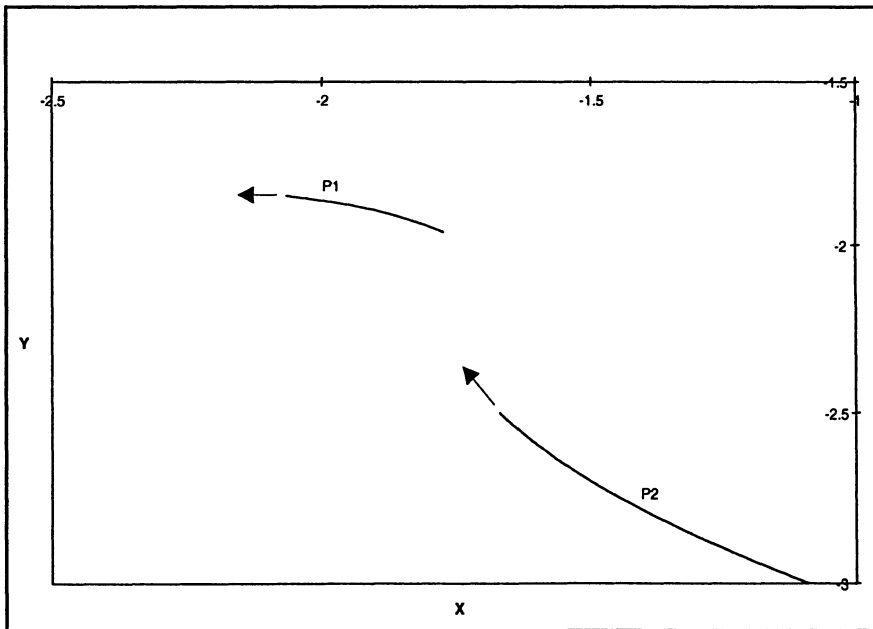


Figure 15.13 Solution Trajectory of Neck Hole Curve Defining Points

15.5 DISCUSSION

In this chapter, we demonstrated the methodology to allow whole-curve constraints in a variational design system. Relations involving curves are represented as a nonlinear optimization problem. In such cases, we analyze the necessary conditions of the NLP. Constraints on the intrinsic behavior of curves are typically represented as integrals over the curve. In these cases, the integral is approximated using quadrature techniques. After all constraints have been entered, the remaining degrees of freedom are used to fair the curve. Curve fairing is also done through a NLP, whose system of necessary conditions forms the final system of equations, the solution set of which contains the desired optimum. The COAST method for design consistency was utilized to maintain a consistent solution once a correct solution had been found. Finally, we demonstrated these techniques both on elementary curves and in an apparel design system.

Using this methodology, parametric objects can be fully incorporated in a variational design system and constrained like any other primitive objects. Relations (e.g., distance and angle) can be entered by the designer between any primitive objects (points, lines, circles, and arcs) as well as curves. Intrinsic properties of a curve (e.g., arc length and total curvature) can also be constrained by the designer in a similar manner to constraining the length of a line segment. Design consistency is maintained whenever the designer modifies the constraints.

REFERENCES

1. Robert Light and David Gossard, "Modification of Geometric Models through Variational Geometry", *Computer-Aided Design*, vol. 14, no. 4, pp 209-214, July 1982.
2. V.C. Lin, D.C. Gossard, and R.A. Light, "Variational Geometry in Computer-Aided Design", *ACM Transactions on Computer Graphics*, vol. 15, no. 3, pp 171-177, August 1981.
3. S. Alasdair Buchanan and Alan de Pennington, "Constraint Definition System: A Computer Algebra Based Approach to Solving Geometric Problems", *Computer Aided Design*, December, vol. 25, no. 12, pp. 741-750, 1993.
4. Horst Nowacki and Xinmin Lu, "Fairing Composite Polynomial Curves with Constraints", *Computer Aided Geometric Design*, vol. 11, pp. 1-15, 1994.
5. John A. Roulier, "Specifying the Arc Length of Bezier Curves", *Computer Aided Geometric Design*, vol 10, pp 25-56, 1993.
6. Michel Bercovier and Arie Jacobi, "Minimization, Constraints, and Composite Bezier Curves", *Computer-Aided Geometric Design*, vol. 11, pp 533-563, 1994.
7. Hans Hagen and Georges-Pierre Bonneau, "Variational Design of Smooth Rational Bezier Curves", *Computer-Aided Geometric Design*, vol. 8, pp 393-399, 1991.
8. Barry Fowler and Richard Bartels, "Constraint-Based Curve Manipulation", *IEEE Computer Graphics and Applications*, pp. 43-49, September 1993.
9. John A. Gregory and Muhammad Sarfraz, "Interactive Curve Design using C2 Rational Splines", *Computers and Graphics*, vol. 18, no. 2, pp 153-159, 1994.
10. Wieger Wesselink and Remco C. Veltcamp, "Interactive Design of Constrained Variational Curves", *Computer Aided Geometric Design*, vol. 12, pp 533-546, 1995.
11. William H. Press et al., *Numerical Recipes: The Art of Scientific Computing, 2nd Edition*, Cambridge University Press, 1992.
12. Arnold Neumaier, "Rigorous Sensitivity Analysis for Parameter-Dependent Systems of Equations", *Journal of Mathematical Analysis and Applications*, vol. 144, pp 16-25, 1989.
13. Helen J. Armstrong, *Patternmaking for Fashion Design*, Harper & Row, New York, 1987.

CHAPTER 16

CREATING A CONSISTENT 3-D VIRTUAL LAST FOR PROBLEMS IN THE SHOE INDUSTRY

In the shoe design industry, there are hundreds of standard lasts (3-D models of a foot) available [2, 3]. Shoe companies select specific lasts to serve as their models for specific sizes. Computer-aided design software for the shoe industry, then, is able to take a design (drawn on a 2-D flattened approximation of the last) and grade (scale) it to any of the numerous standard lasts. They are not able, however, to create a new pattern from an arbitrary set of measurements. In shoe design, the ability to quickly and easily customize each pair of shoes for an individual customer could be turned into a large market advantage. Many aspects of the required technology are already available (e.g., information systems, CNC machines). The problem is then reduced to one of computing the desired shoe pattern, given a person's foot measurements and a similar prototype, such that the new pattern satisfies the measurements, but is still similar to the prototype. In such a computerized domain, there is a need for a virtual model of a last that can adjust to any desired size. In this chapter, an approach for creating a virtual last is demonstrated in which a series of 22 3-D Bezier curves are constrained to satisfy a set of measurements given by the designer. As there are typically numerous curves which can satisfy a given system of constraints, the COAST methodology for design consistency developed in Chapters 14 and 15 is implemented to ensure convergence to the intended solution.

16.1 PROBLEMS IN SHOE DESIGN INDUSTRY

“Mass customization gives individual customers exactly what they want, without creating overwhelming choice complexity of pushing costs up to the point that a company prices itself out of the market [1].” Numerous products exist in which the ability to quickly and easily customize each item for an individual customer could be turned into a large market advantage (e.g., computers, clothing, and shoes).

In the case of custom shoe design, the problem is inherently a continuous problem: everybody wants shoes that fit and measurements are continuous. Current industrial solutions, however, force discreteness, minimizing the number of measurements and rounding them off to the nearest increment so as to limit the number of possible designs. The state of manufacturing technology is such that fast information servers (for easy transmission of personal data) and flexible automation

machines (CNC programs to cut the upper patterns) are available. The problem is then essentially reduced to one of computing the desired pattern, given a person's foot measurements and a similar prototype, such that the new pattern satisfies the measurements, but is still similar to the prototype.

This problem can be generalized in either the discrete case or the continuous case. In the discrete case (best fit selection), a shoe manufacturer has a large, but fixed number of sizes available for every different style. More 'custom' shoes can be approximated by using more measurements and allowing more precise increments of the measurements. Additionally, the customer could specify the fit and/or the performance of the shoe (sporty or comfortable). From these measurements and desired behaviors of the shoe, a virtual inventory of lasts could be compared and the best fitting last selected. An outline of this process is shown in Figure 16.1.

In the continuous case (pattern grading/visualization) each shoe is sized to fit the foot exactly. In the shoe design industry, there are many standard lasts (3-D models of a foot) available [2, 3]. Shoe design software, then, is able to take a design and grade it to specific sizes. Shoe design software, then, is able to take a design and grade it to any of the numerous standard sizes. They are not able, however, to create a new pattern from an arbitrary set of measurements.

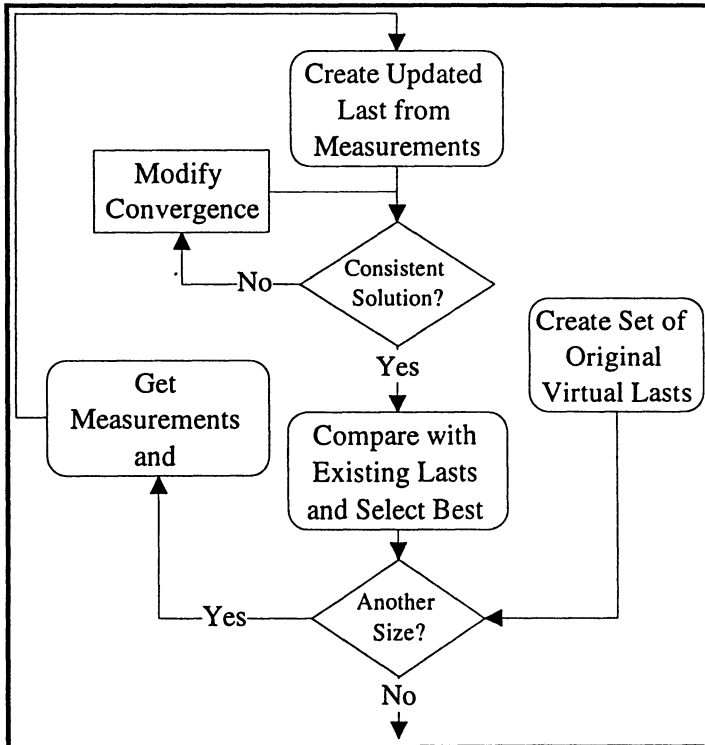


Figure 16.1 Best Fit Selection

In contrast, consider a system that is able to both grade an upper design to any size (determined by user's measurements) as well as render the corresponding design. One such system could work as outlined in Figure 16.2. From a computerized model of a last and its flattened approximation, the user draws the outlines of the desired upper patterns. Given the one-to-one correspondence between the curves on the last and the flattened approximation, any point selected on the flattened last can be identified on the 3-D last. When the measurements are modified, the last and the flattened approximation are consistently updated and the points which make up the upper patterns are automatically adjusted.

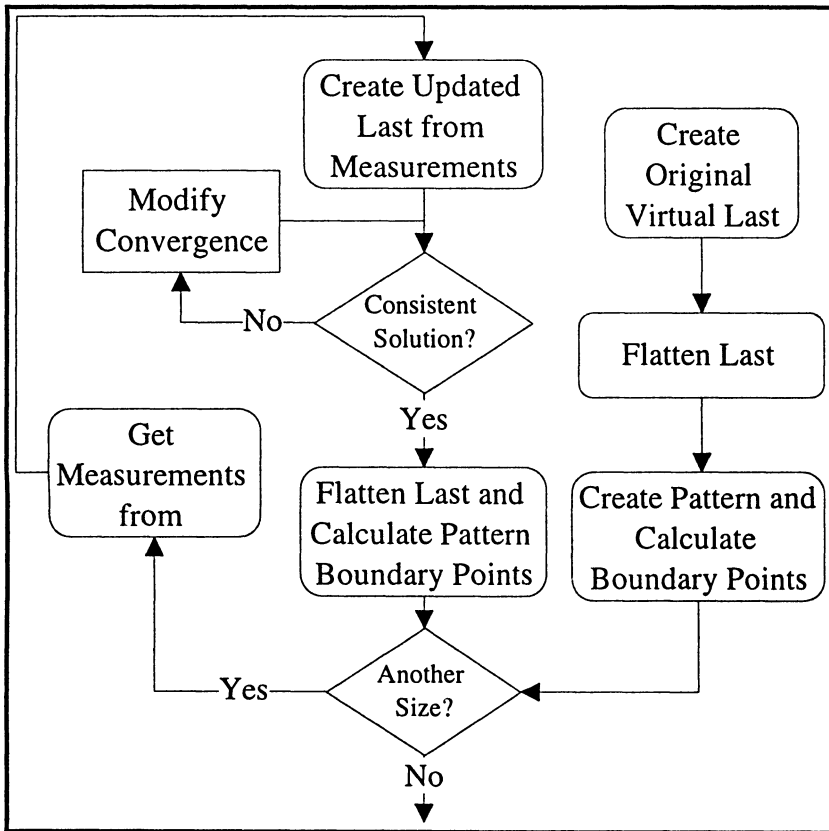


Figure 16.2 Upper Pattern Grading and Visualization

In each case, there is a need for a consistent 3-D virtual last to optimally determine issues of size and fit. Given a set of measurements, this chapter demonstrates the ability to create a set of curves which outline a 3-D virtual last. Constraints on the curves are transformed into nonlinear equations. The remaining degrees of freedom are constrained by fairing the curves. Design consistency is

maintained by tracking the desired local optimum of the NLP using the COAST methodology (see Chapters 14 and 15) rather than finding the global optimum.

16.2 CREATION OF A VIRTUAL LAST

Our system for creating a parametrized 3-D last, DANSER (Design Algorithms using Nonlinear Systems of Equations Relationally), uses four sets of Bezier curves to describe a foot or a last (see Figures 16.3-16.6, respectively):

1. Nine curves that surround the outside lower part of the foot ($\underline{P}_1 - \underline{P}_9$);
2. Five curves that surround the outside upper part of the foot ($\underline{P}_{10} - \underline{P}_{14}$);
3. Five curves that pass between the first two sets of curves ($\underline{P}_{15} - \underline{P}_{19}$); and
4. Three curves that pass under the foot ($\underline{P}_{20} - \underline{P}_{22}$).

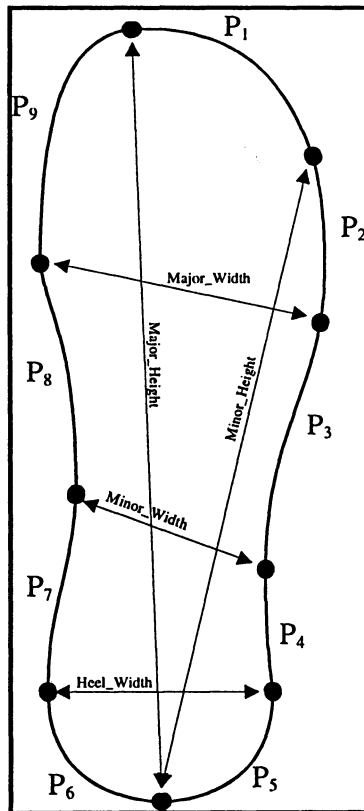


Figure 16.3 Constraint Points of Lower Curve in XY Plane

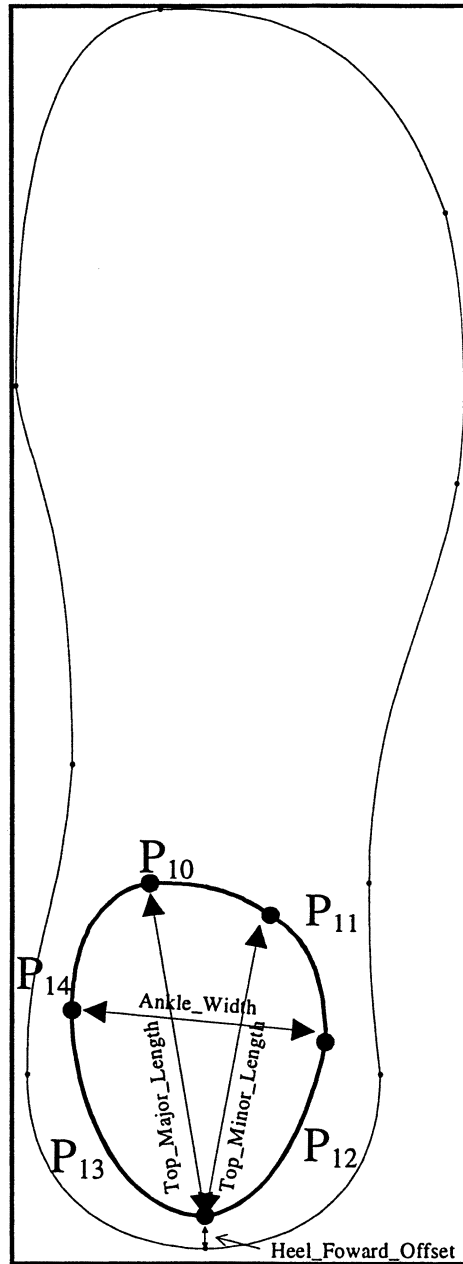


Figure 16.4 Constraint Points of Upper Curve in XY Plane

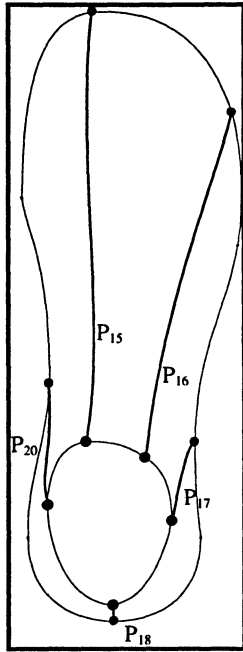


Figure 16.5 Constraint Points of Connecting Curves in XY Plane

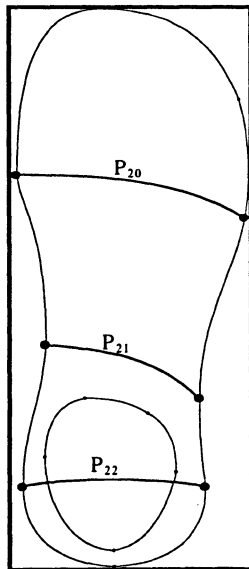


Figure 16.6 Constraint Points of Bottom Curves in XY Plane

Table 16.1 User-Defined Measurements of A Foot

Measurement	M-11	W-9	Measurement	M-11	W-9
<i>Top_X</i>	-1.0	-1.0	<i>Top_Y</i>	10.45	9.45
<i>Left_X</i>	-2.0	-2.0	<i>Left_Y</i>	7.25	7.25
<i>Right_X</i>	2.25	1.7	<i>Right_Y</i>	6.4	6.4
<i>Minor_Length</i>	9.0	7.5	<i>Minor_Width</i>	3.0	2.75
<i>Heel_Width</i>	3.0	2.25	<i>Ankle_Width</i>	2.5	2.25
<i>Top_Major_Length</i>	3.5	3.25	<i>Top_Minor_Length</i>	3.0	2.75
<i>Heel_Forward_Offset</i>	0.25	0.25	$\underline{P}_{6z}(0)$	1.0	0.75
$L(\underline{P}_1)$	4.0	3.5	$L(\underline{P}_2)$	2.0	2.0
$L(\underline{P}_3)$	3.5	3.25	$L(\underline{P}_4)$	2.0	1.5
$L(\underline{P}_5)$	2.0	1.5	$L(\underline{P}_6)$	2.0	1.5
$L(\underline{P}_7)$	3.5	2.5	$L(\underline{P}_8)$	3.5	3.25
$L(\underline{P}_9)$	3.0	2.75	$L(\underline{P}_{10})$	1.5	1.25
$L(\underline{P}_{11})$	2.5	2.25	$L(\underline{P}_{12})$	2.5	2.0
$L(\underline{P}_{13})$	3.0	3.25	$L(\underline{P}_{14})$	2.0	1.75
$L(\underline{P}_{15})$	6.75	7.25	$L(\underline{P}_{16})$	6.0	6.0
$L(\underline{P}_{17})$	2.0	2.0	$L(\underline{P}_{18})$	1.25	1.5
$L(\underline{P}_{19})$	1.75	1.75	$L(\underline{P}_{20})$	5.25	4.25
$L(\underline{P}_{21})$	4.25	3.75	$L(\underline{P}_{22})$	3.75	3.25
$\underline{P}_{1z}(0)$.5	.4	$\underline{P}_{2z}(0)$.33	.25
$\underline{P}_{3z}(0)$.5	.5	$\underline{P}_{4z}(0)$	0.75	.5
$\underline{P}_{5z}(0)$	0.75	.75	$\underline{P}_{7z}(0)$.75	.75
$\underline{P}_{8z}(0)$	2.0	1.25	$\underline{P}_{9z}(0)$	0.5	0.5
$\underline{P}_{10z}(0)$	3.5	3.25	$\underline{P}_{11z}(0)$	3.25	2.75
$\underline{P}_{12z}(0)$	2.5	2.25	$\underline{P}_{13z}(0)$	2.25	2.25
$\underline{P}_{14z}(0)$	3.0	2.25	$dist(\underline{P}_{15}, \underline{P}_{20})$	1.5	1.5
$dist(\underline{P}_{15}, \underline{P}_{21})$	2.25	2.0	$dist(\underline{P}_{15}, \underline{P}_{22})$	4.5	3.75

16.2.1 CONSTRAINT DEFINITIONS

The 3-D last is dimensioned to 52 user-defined measurements identified in Table 16.1 (along with their values for a men's size 11 and women's size 9 foot). The measurements include thickness and width dimensions measured at peaks and valleys of the foot as well as measurements of distance along the foot (designated by $L(P_i)$) and height measurements (designated by $\underline{P}_{i_z}(0) = \underline{P}_{i_z}(\underline{G}_i 0)$).

The 22 curves have a total of 264 unknowns (270 after including the additional curve parameters for the distance relations). From the measurements, a system of 227 constraints is formed. The 22 arc length measurements, the 14 height measurements, and the 3 distance measurements (combined with the six additional necessary conditions from the distance constraints) given in Table 16.1 yield 45 constraints.

The parametric function for cubic Bezier curves is:

$$P_i(\underline{G}_i, s_i) = \begin{bmatrix} P_{i,x}(\underline{G}_i, s_i) \\ P_{i,y}(\underline{G}_i, s_i) \\ P_{i,z}(\underline{G}_i, s_i) \end{bmatrix} =$$

$$(1-s_i)^3 P_{i,0} + 3(1-s_i)^2 s_i P_{i,1} + 3(1-s_i) s_i^2 P_{i,2} + s_i^3 P_{i,3} \text{ where}$$

$$\underline{G}_i = \left[P_{i,0_x}, P_{i,1_x}, P_{i,2_x}, P_{i,3_x}, P_{i,0_y}, P_{i,1_y}, P_{i,2_y}, P_{i,3_y}, P_{i,0_z}, P_{i,1_z}, P_{i,2_z}, P_{i,3_z} \right] \text{ and}$$

$0 \leq s_i \leq 1$. Each distance constraint between two curves, P_i and P_j , is of the form:

$$\sqrt{\left(P_{i,x}(\underline{G}_i, s_i) - P_{j,x}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,y}(\underline{G}_i, s_i) - P_{j,y}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,z}(\underline{G}_i, s_i) - P_{j,z}(\underline{G}_j, s_j) \right)^2} = d$$

and the two additional necessary conditions for each distance constraint are:

$$\frac{\partial}{\partial s_i} \sqrt{\left(P_{i,x}(\underline{G}_i, s_i) - P_{j,x}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,y}(\underline{G}_i, s_i) - P_{j,y}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,z}(\underline{G}_i, s_i) - P_{j,z}(\underline{G}_j, s_j) \right)^2} = 0 \text{ and}$$

$$\frac{\partial}{\partial s_j} \sqrt{\frac{\left(P_{i,x}(\underline{G}_i, s_i) - P_{j,x}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,y}(\underline{G}_i, s_i) - P_{j,y}(\underline{G}_j, s_j) \right)^2 + \left(P_{i,z}(\underline{G}_i, s_i) - P_{j,z}(\underline{G}_j, s_j) \right)^2}{\partial s_j}} = 0, \text{ adding two}$$

equations and two unknowns (the parameter values). The arc length constraint of any given curve, P_i , was calculated using the integral formula,

$$\int_{s_i=0}^1 \sqrt{\left[\frac{\partial(P_{i,x}(\underline{G}_i, s_i))}{\partial s_i} \right]^2 + \left[\frac{\partial(P_{i,y}(\underline{G}_i, s_i))}{\partial s_i} \right]^2 + \left[\frac{\partial(P_{i,z}(\underline{G}_i, s_i))}{\partial s_i} \right]^2} ds, \text{ and estimated}$$

using a four-point quadrature technique.

The remaining 182 constraints are divided among connectivity constraints, gradient constraints, distance constraints, and other explicit constraints as follows:

Connectivity Constraints

$\underline{P}_1(1) = \underline{P}_2(0)$	$\underline{P}_2(1) = \underline{P}_3(0)$	$\underline{P}_3(1) = \underline{P}_4(0)$	$\underline{P}_4(1) = \underline{P}_5(0)$
$\underline{P}_5(1) = \underline{P}_6(0)$	$\underline{P}_6(1) = \underline{P}_7(0)$	$\underline{P}_7(1) = \underline{P}_8(0)$	$\underline{P}_8(1) = \underline{P}_9(0)$
$\underline{P}_9(1) = \underline{P}_1(0)$			
$\underline{P}_{10}(1) = \underline{P}_{11}(0)$	$\underline{P}_{11}(1) = \underline{P}_{12}(0)$	$\underline{P}_{12}(1) = \underline{P}_{13}(0)$	$\underline{P}_{13}(1) = \underline{P}_{14}(0)$
$\underline{P}_{14}(1) = \underline{P}_{10}(0)$			
$\underline{P}_{15}(1) = \underline{P}_1(0)$	$\underline{P}_{15}(0) = \underline{P}_{10}(0)$	$\underline{P}_{16}(1) = \underline{P}_2(0)$	$\underline{P}_{16}(0) = \underline{P}_{11}(0)$
$\underline{P}_{17}(1) = \underline{P}_4(0)$	$\underline{P}_{17}(0) = \underline{P}_{12}(0)$	$\underline{P}_{18}(1) = \underline{P}_6(0)$	$\underline{P}_{18}(0) = \underline{P}_{13}(0)$
$\underline{P}_{19}(1) = \underline{P}_8(0)$	$\underline{P}_{19}(0) = \underline{P}_{14}(0)$		
$\underline{P}_{20}(1) = \underline{P}_3(0)$	$\underline{P}_{20}(0) = \underline{P}_9(0)$	$\underline{P}_{21}(1) = \underline{P}_4(0)$	$\underline{P}_{21}(0) = \underline{P}_8(0)$
$\underline{P}_{22}(1) = \underline{P}_5(0)$	$\underline{P}_{22}(0) = \underline{P}_7(0)$		

Each connectivity constraint adds one equation for each dimension. For example, the

first connectivity constraint adds three equations:

$$P_{1,3_x} - P_{2,0_x} = 0, \quad P_{1,3_y} - P_{2,0_y} = 0, \quad \text{and} \quad P_{1,3_z} - P_{2,0_z} = 0.$$

The x-, y-, and z-coordinates are constrained at each point for a total of $30 \times 3 = 90$ connectivity constraints.

Gradient Constraints

$$\begin{aligned} \partial \underline{P}_1(1) = \partial \underline{P}_2(0) & \quad \partial \underline{P}_2(1) = \partial \underline{P}_3(0) & \quad \partial \underline{P}_3(1) = \partial \underline{P}_4(0) & \quad \partial \underline{P}_4(1) = \partial \underline{P}_5(0) \\ \partial \underline{P}_5(1) = \partial \underline{P}_6(0) & \quad \partial \underline{P}_6(1) = \partial \underline{P}_7(0) & \quad \partial \underline{P}_7(1) = \partial \underline{P}_8(0) & \quad \partial \underline{P}_8(1) = \partial \underline{P}_9(0) \\ \partial \underline{P}_9(1) = \partial \underline{P}_1(0) & & & \end{aligned}$$

$$\partial \underline{P}_{10}(1) = \partial \underline{P}_{11}(0) \quad \partial \underline{P}_{11}(1) = \partial \underline{P}_{12}(0) \quad \partial \underline{P}_{12}(1) = \partial \underline{P}_{13}(0) \quad \partial \underline{P}_{13}(1) = \partial \underline{P}_{14}(0)$$

$$\partial \underline{P}_{14}(1) = \partial \underline{P}_{10}(0)$$

$$\partial \underline{P}_3(0) / \partial x = 0 \quad \partial \underline{P}_4(0) / \partial x = 0 \quad \partial \underline{P}_5(0) / \partial x = 0 \quad \partial \underline{P}_6(0) / \partial y = 0$$

$$\partial \underline{P}_7(0) / \partial x = 0 \quad \partial \underline{P}_8(0) / \partial x = 0 \quad \partial \underline{P}_8(0) / \partial z = 0 \quad \partial \underline{P}_9(0) / \partial x = 0$$

$$\partial \underline{P}_{10}(0) / \partial y = 0 \quad \partial \underline{P}_{12}(0) / \partial x = 0 \quad \partial \underline{P}_{13}(0) / \partial y = 0 \quad \partial \underline{P}_{14}(0) / \partial x = 0$$

$$\partial \underline{P}_{15}(1) / \partial x = 0 \quad \partial \underline{P}_{15}(1) / \partial y = 0 \quad \partial \underline{P}_{16}(1) / \partial x = 0 \quad \partial \underline{P}_{16}(1) / \partial y = 0$$

$$\partial \underline{P}_{17}(1) / \partial x = 0 \quad \partial \underline{P}_{17}(1) / \partial y = 0 \quad \partial \underline{P}_{18}(1) / \partial x = 0 \quad \partial \underline{P}_{18}(1) / \partial y = 0$$

$$\partial \underline{P}_{19}(1) / \partial x = 0 \quad \partial \underline{P}_{19}(1) / \partial y = 0$$

$$\partial \underline{P}_{20}(0) / \partial x = 0 \quad \partial \underline{P}_{20}(0) / \partial y = 0 \quad \partial \underline{P}_{20}(1) / \partial x = 0 \quad \partial \underline{P}_{20}(1) / \partial y = 0$$

$$\partial \underline{P}_{21}(0) / \partial x = 0 \quad \partial \underline{P}_{21}(0) / \partial y = 0 \quad \partial \underline{P}_{21}(1) / \partial x = 0 \quad \partial \underline{P}_{21}(1) / \partial y = 0$$

$$\partial \underline{P}_{22}(0) / \partial x = 0 \quad \partial \underline{P}_{22}(0) / \partial y = 0 \quad \partial \underline{P}_{22}(1) / \partial x = 0 \quad \partial \underline{P}_{22}(1) / \partial y = 0$$

Each gradient constraint adds one equation for each dimension. For example, the first gradient constraint adds three equations:

$$-3P_{1,2_x} + 3P_{1,3_x} - 3P_{2,0_x} + 3P_{2,1_x} = 0, \quad -3P_{1,2_y} + 3P_{1,3_y} - 3P_{2,0_y} + 3P_{2,1_y} = 0,$$

$$\text{and } -3P_{1,2_z} + 3P_{1,3_z} - 3P_{2,0_z} + 3P_{2,1_z} = 0$$

There are 14 points at which all three gradients are constrained ($\frac{\partial x}{\partial t}$, $\frac{\partial y}{\partial t}$, and $\frac{\partial z}{\partial t}$) and 34 points where individual gradients are explicitly constraints for a total of 76 gradient constraints.

Distance Constraints

An additional six constraints are formed from the remaining measurements in Table 16.1:

$$\begin{aligned} \text{dist}(\underline{P}_2(0), \underline{P}_6(0)) &= \text{Minor_Length} & \text{dist}(\underline{P}_4(0), \underline{P}_8(0)) &= \text{Minor_Width} \\ \text{dist}(\underline{P}_5(0), \underline{P}_7(0)) &= \text{Heel_Width} & \text{dist}(\underline{P}_{12}(0), \underline{P}_{14}(0)) &= \text{Ankle_Width} \\ \text{dist}(\underline{P}_{10}(0), \underline{P}_{13}(0)) &= \text{Top_Major_Length} & \text{dist}(\underline{P}_{11}(0), \underline{P}_{13}(0)) &= \text{Top_Minor_Length} \end{aligned}$$

Each distant constraint adds one equation. For example, the first distance constraint adds the equation:

$$\sqrt{\left(P_{2,0_x} - P_{6,0_x}\right)^2 + \left(P_{2,0_y} - P_{6,0_y}\right)^2 + \left(P_{2,0_z} - P_{6,0_z}\right)^2} = \text{Minor_Length} .$$

Explicit Constraints

Finally, ten explicit constraints (similar in form to the connectivity constraints) are used to orient the last:

$$\begin{aligned} \underline{P}_{6_x}(0) &= 0 & \underline{P}_{6_y}(0) &= 0 & \underline{P}_{1_x}(0) &= \text{Top_X} \\ \underline{P}_{1_y}(0) &= \text{Top_Y} & \underline{P}_{3_x}(0) &= \text{Right_X} & \underline{P}_{3_y}(0) &= \text{Right_Y} \\ \underline{P}_{9_x}(0) &= \text{Left_x} & \underline{P}_{9_y}(0) &= \text{Left_Y} & \underline{P}_{13_x}(0) &= 0 \\ \underline{P}_{13_y}(0) &= \text{Heel_Forward_Offset} & & & & \end{aligned}$$

16.2.2 CURVE FAIRING

The remaining degrees of freedom were constrained by minimizing the sum of the

stiffness of the curves as given by the formula: $\frac{1}{2} \cdot \int (\|P''(\underline{G}, s)\|^2 + \|P'''(\underline{G}, s)\|^2) ds$

which is then approximated by a four-point quadrature technique. In order to maintain the intended solution, the local optimum was maintained by following the solution to the necessary conditions of the optimization problem as the measurements were incrementally moved towards their updated values. The resulting system had 497 equations and 497 unknowns.

16.3 RESULTS

The measurements for a men's size 11 were entered and the corresponding 264 parameters for the 22 Bezier curves were calculated (shown in Figures 16.7-A and 16.7-B).

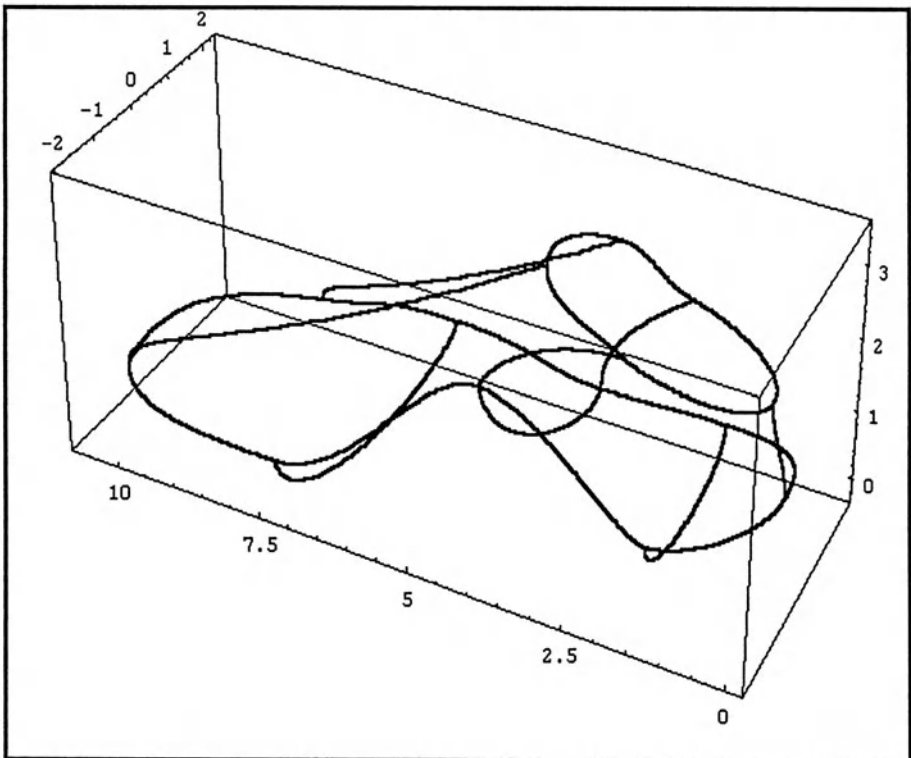


Figure 16.7-A Parameterized Last - Men's Size 11

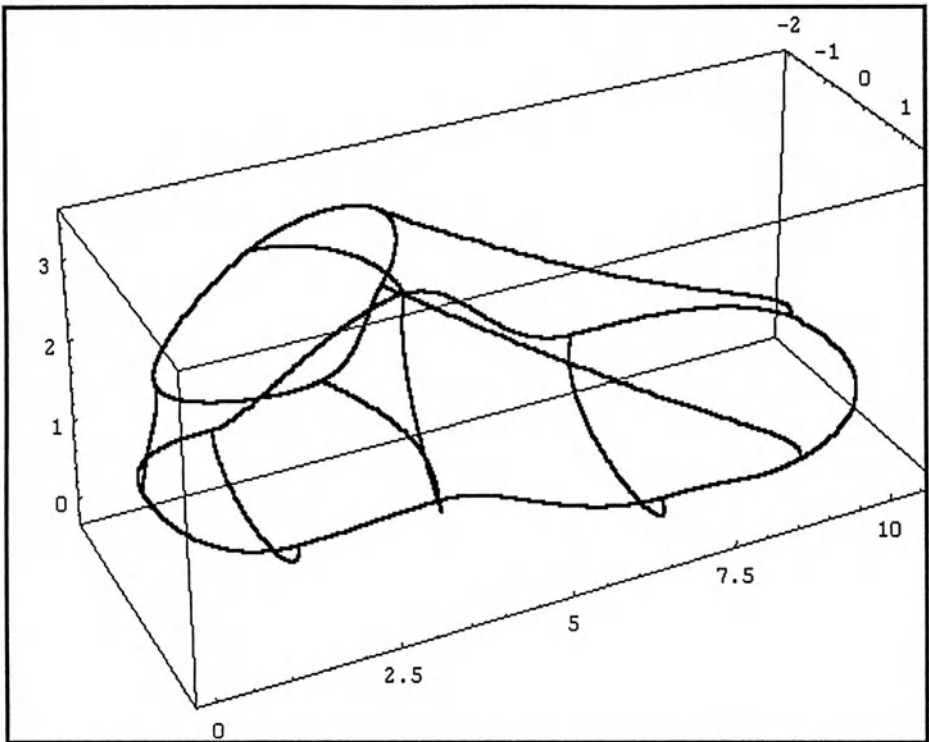


Figure 16.7-B Parameterized Last - Men's Size 11

Using the COAST presented in Chapters 14 and 15, the current solution was then tracked when the measurements were modified to those of a women's size 9 in order to maintain a consistent model of the foot (Figures 16.8-A and 16.8-B). This is in contrast to an inconsistent solution (Figure 16.8-C, for example). In this configuration, the curve that extends over the little toe actually starts with a negative z-gradient. This curve satisfies all the constraints, yet is obviously not the desired solution.

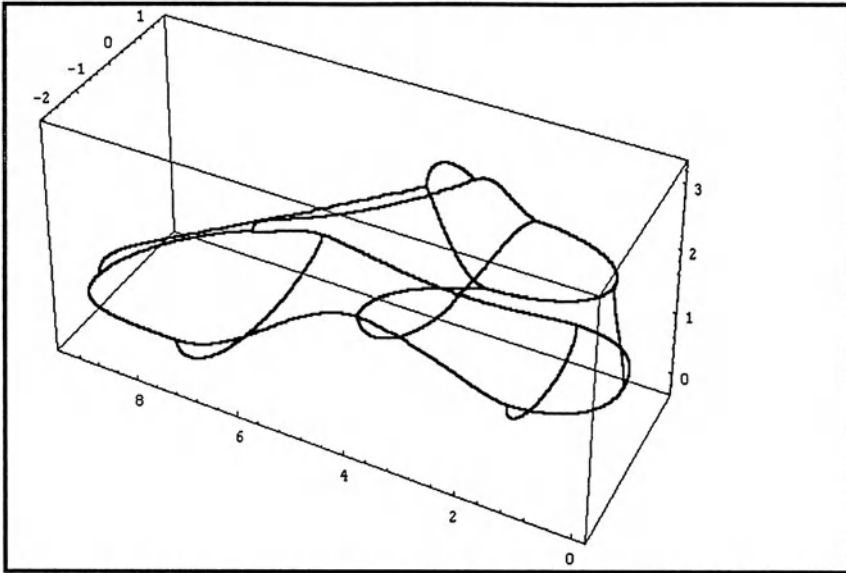


Figure 16.8-A Parameterized Last - Women's Size 9

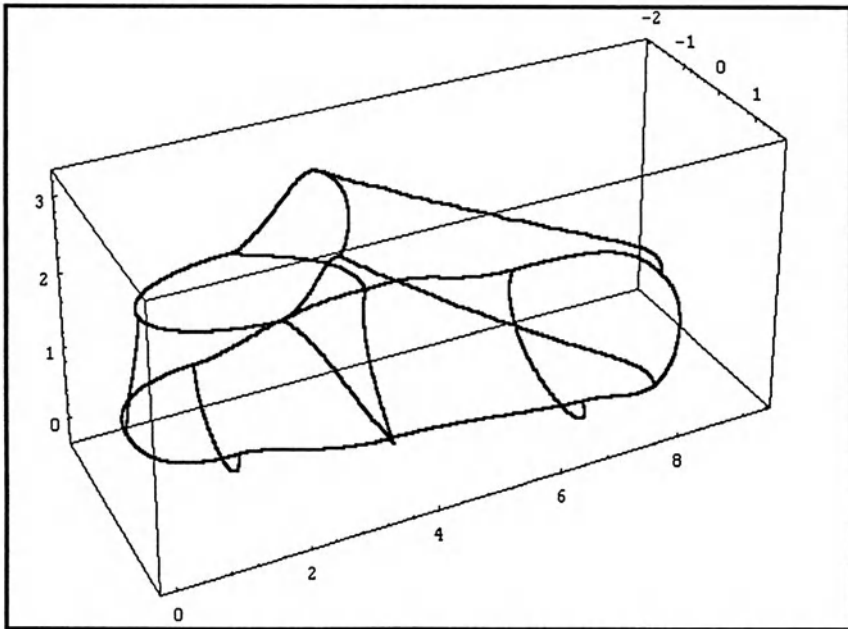


Figure 16.8-B Parameterized Last - Women's Size 9

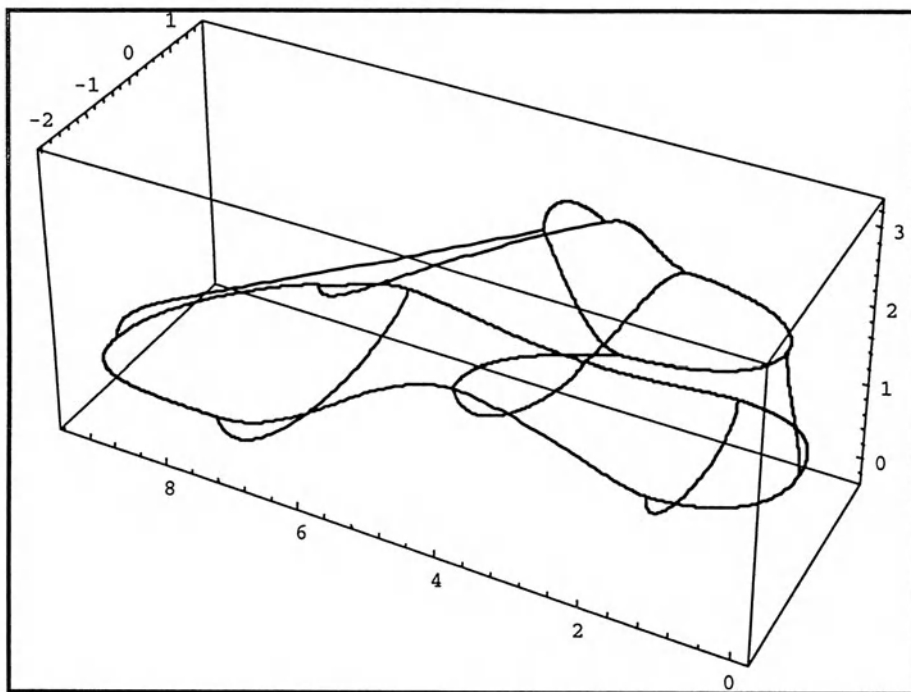


Figure 16.8-C Inconsistent Parameterized Last - Women's Size 9

16.4 SUMMARY

In this chapter, we have demonstrated the ability to create a 3-D last, composed of a series of Bezier curves, that is constrained to achieve user-defined measurements. Every constraint on the curves is transformed into an algebraic equation. Any remaining degrees of freedom are faired using the stiffness criterion. In order to maintain a consistent solution, the local optima is followed using the necessary conditions of the NLP. When any of the measurements are modified, DANSER is able to update the last and maintain a consistent solution. Currently, the last is parametrized with 52 measurements which is a rather large number. In the future, this number can be reduced with additional study, by finding additional correlation between certain measurements and aspects of the shape of the foot.

REFERENCES

1. New York Times, March 20, 1996, Section D, page 1.
2. Karl C. Adrian, *American Last Making*, Shoe Trades Publishing, Massachusetts, 1991.
3. Michael H. Sharp, *The Pattern Cutter's Handbook*, Footwear OPEN TECH Unit, England, 1994.

PART FOUR

DETAILED DESIGN APPLICATIONS

CHAPTER 17

DESIGN OF A WORMGEAR REDUCER: A CASE STUDY

17.1 INTRODUCTION

This chapter presents a case study of a wormgear reducer design, which is used to corroborate the evolutionary model of the design process (Chapter 6). The design of the wormgear reducer is performed on both conceptual and detailed (parametric) levels. Conceptualization mainly involves inventive activities in the form of the generation of alternative possible solutions to the required goals (high level specifications). In parametric design, the dimensions of a part are calculated by solving a system of constraints (typically nonlinear equations). The constraints are considered part of the specifications. The following sections focus mainly on the parametric design of the wormgear reducer. Let us recapitulate some of the very basic features of the evolutionary model as articulated for parametric design.

- An artifact (design), at any particular abstraction level, is described in terms of part types (a group of objects which are similar but have different sizes). Every part can be described by a set of attributes. Each attribute can be described by its dimension (such as wire diameter, spring diameter, number of active coils, and modulus of elasticity).
- Specifications or constraints, at any particular abstraction level, are the various functional, behavioral, performance, reliability, aesthetic, or other characteristics or features that are to be present in the physically implemented artifact. In parametric design, closed-form constraints are usually either *Euclidean* (including distance, tangency, parallelism, and so on), or *functional* (such as mass properties, forces, stiffness, strength, rating life, and so on). A higher order constraint is a property that is satisfied by lower order constraints.
- Design proceeds as a succession of cycles. In each cycle, the design (or 'design parameters') is tested against the specifications (or 'constraints').
- If the design satisfies the specifications, then there is a fit between the two, otherwise there is said to be a misfit between design and specifications.
- Depending on the source of the misfit, the design and/or the specifications are

modified to eliminate or reduce the misfit, thus producing a new (design, specifications) pair.

- The design process halts when adaptation is achieved. However, the process may resume if the specifications are subsequently altered, in which case the previous state of adaptation is disturbed.
- The process uses only the facts, rules, and laws included in the designer's knowledge body.

Limitation of space and readability force one to limit the size of the case study, and using prose descriptions. We associate to each item in the text a label, denoting whether it is a constraint (C), design parameter (DP), or a part of the knowledge body (KB). It would be clear to the reader that dependencies between constraints and design parameters (which produce a new pair) can be represented by rules given either in the form of equations or in the form of a directed acyclic graph (if-then structures).

17.2 CONCEPTUAL DESIGN OF A WORMGEAR REDUCER (GEAR BOX)

17.2.1 CONFRONTATION

Consider you are an engineer for Sears & Roebuck Co. The director of the Product Development Department ask you to design a mass production device (C_1) which can be used for lifting light objects (C_2) or opening a garage door in a family or small warehouse (C_3).

The specifications for the device are not clear enough for you to start the design. You must talk with the Director for more details. According to your experience (KB), the "device" he asked is in fact called Hoist (DP_1). Unlike hoists, used in industries and construction sites (KB), this one should be small (C_4), light (C_5), and at an affordable low cost (C_6). Considering most moving and loading tasks in families or small warehouses (KB), the capacity needed for such a hoist is in the rage of 100 to 150 lb (C_7).

Considering most hoists used in industries and construction sites, the hoist should include a cable (DP_2), drum (DP_3), coupling (DP_4), brake (DP_5), reducer (DP_6) and driven by a crane motor (DP_7). The working principle of the hoist is shown in Figure 17.1.

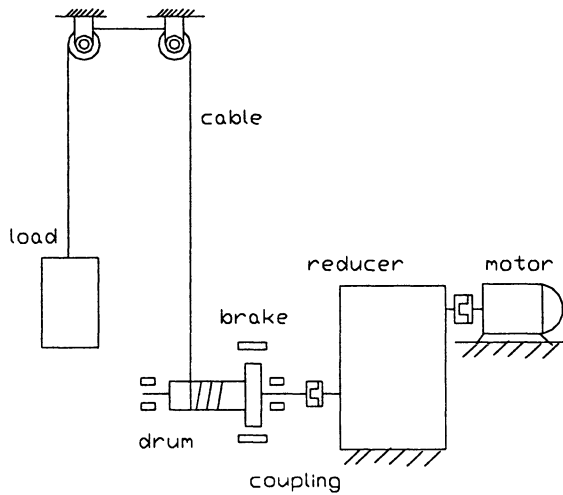


Figure 17.1 The Working Principle of the Hoist

17.2.2 PROBLEM FORMULATION

Suppose we are going to select the drum, cable, and electrical motor directly from the market. For general usage (in small warehouse), the lifting speed shouldn't be too fast ($V = 30$ feet per hour would be a suitable speed, C_8). To raise a load of $F = 150$ lb. (C_9), a wire rope must have at least 10 times of the load 150 lb., e.g. 1,500 lb (C_{10}). From the catalog, we select 1/4 inch diameter (DP_8) wire rope (breaking strength 4,000 lb). To insure a long work life for the rope (C_{11}), the overbending of the rope (C_{12}) must be avoided. Thus, the minimum drum diameter should be large enough (C_{13}). According to the engineering recommendation (KB) value (15 to 20 times the rope diameter), the designer chooses the drum diameter $d_{dr} = 3.75$ inches (DP_9). Most industrial electric motors have a synchronous (no load) speed of $n_m = 1,500$ rpm (DP_{10}). The basic requirement of designing a reducer is replaced with the following specifications and constraints:

Input speed: $n_i = 1500rpm$ (motor speed) (1)

Output reducer speed (C_{14}): $n_o = \frac{12V}{\pi d_{dr}} = \frac{12 \cdot 30}{3.14 \cdot 3.75} = 30.57rpm$ (2)

Reduction ratio (C_{15}): $R = \frac{n_i}{n_o} = \frac{1500}{30.57} = 49.067 \approx 50$ (3)

Output power (C_{16}):
$$P_o = \frac{FV}{33,000} = \frac{150 * 30}{33,000} = 0.136(\text{hp}) \quad (4)$$

Duration (C_{17}): 2000 hours per year for 5 years with a reliability of 90%

Overall dimension (C_{18}): 8" x 8" x 8" (L x W x H)

No special material and special machining process should be used.

17.2.3 DESIGN CONCEPTS

There are many different kind of reducers that can be designed. The designer can start drawing some preliminary sketches (conceptualizations) as shown in Figure 17.2. Belts, chains, gears, and worm gears can be considered as the means of transmission.

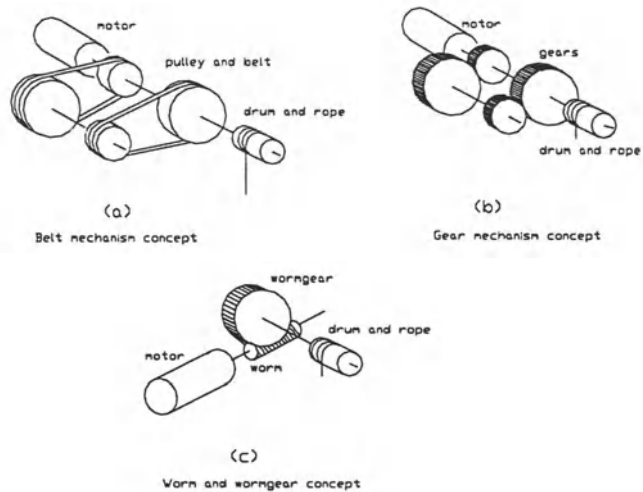


Figure 17.2 Sketch of the Design Concepts

An evaluation operator (see Definition 6.1) is defined and used to measure the degree of efficiency of each alternative. Many criteria have to be considered (associated with each criteria is a weighting factor):

1. Production (including material and machining costs) and operation costs (0.1);
2. Overall dimension and weight (1);
3. Convenience of use and maintenance (0.7);
4. Reliability and duration (0.4);

5. **Simplicity (0.8).** The simplicity measures introduced in Chapter 8 can be applied at this stage.

Table 17.1 shows the rating of each concept with respect to the criteria (lower is better):

Table 17.1 The Rating of the Design Concepts

	V belt & pulley	chain	spur gears	worm & worm gear
complexity of the structure	simple (1)	simple (1)	simple (1)	simple (1)
material	steel (1)	steel (1)	steel (1)	steel and bronze (1)
overall dimension	large (6)	medium (4)	compact (3)	small (1)
transmitting accuracy	low (6)	high (2)	high (2)	high (2)
mechanical efficiency	0.7–0.9 (2)	0.8–0.9 (1)	0.8–0.95 (1)	0.45–0.85 (3)
machining equipment	lathe (1)	lathe, mill (2)	lathe, mill or hob (4)	lathe & hob (2)
machining precision required	low (1)	medium (2)	high (3)	high (3)
assembly difficulty	easy (1)	easy (1)	easy (1)	easy (1)
noise level	low (1)	high (4)	medium (2)	medium (2)
maintenance	change belt (4)	lubrication (6)	check oil (1)	check oil (1)

The belt concept is the simplest transmission mechanism, pulley and shaft can be turned on a lathe. Since the power is transmitted by friction, the ratio is unstable when the load changes. Moreover, certain distance between two wheel centers must be maintained, to insure that the belt cover angle is greater than 120 degree. This makes the design overall dimension rather large.

Spur gear mechanism can obtain accurate transmission ratio, and give more compact design. However, special gear generating machine (such as hob) is needed.

For both belt and gear mechanisms, at least two or three stages are needed to obtain the reduction ratio of 1:50 (C_{15}). It means that more parts will be included in the design. The design would be rather complicated in structure and larger in overall size.

Worm and wormgear mechanism is widely used for high ratio speed change. It can reach the ratio of 30 ~100 in a single stage. Accordingly, comparing these alternatives, we decide to select the worm and wormgear concept (DP_{11}).

The resulting dependencies (up to this stage) between constraints and design

parameters is shown in Figure 17.3.

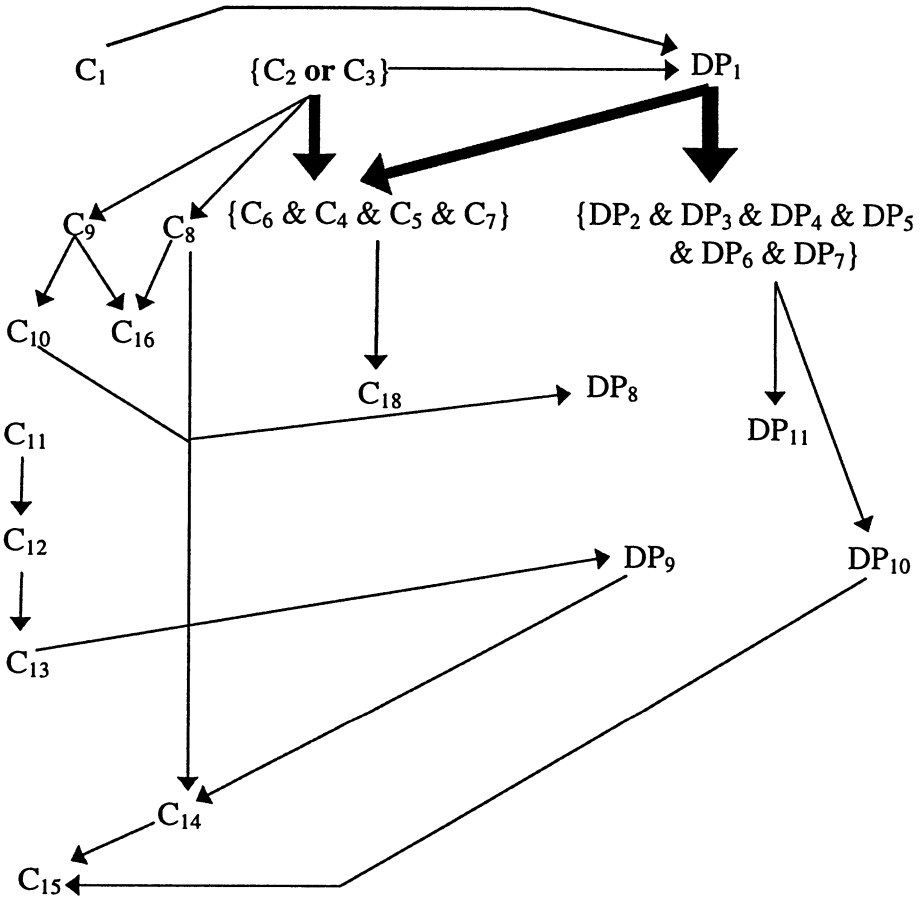


Figure 17.3 A First Snapshot of the Constraints and Design Parameters Graph

17.3 DETAILED SYNTHESIS OF THE GEAR BOX

During this stage, we design the actual components and mechanism including material, shapes, dimensions, and the relative positions of the elements. The specifications that are considered include strength, rigidity, reliability, wear, friction, cost, ratio requirements, and space limitation.

By studying manufacturing catalogs (KB), and understanding the structural features and functions of industrial reducers, the designer arrives at a design plan which consists of designing the structure of the transmission parts (worm and

wormgear), shaft, bearings, casing (box), keys, couples, 'O' rings, screws, and the method of lubricating and sealing. The material, machining and assembling processes are also need to be considered. Figure 17.4 and Figure 17.5 show a typical worm and wormgear reducer and its inside structure.

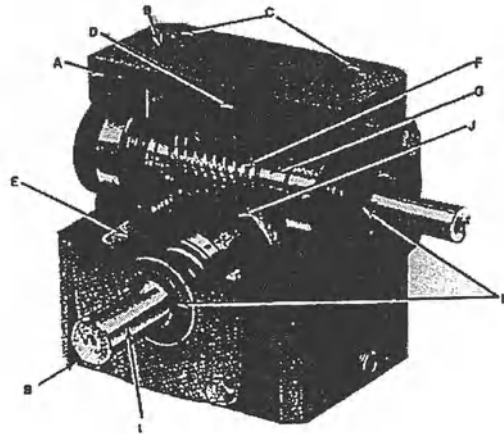


Figure 17.4 A Worm and Wormgear Reducer from BOSTON GEAR

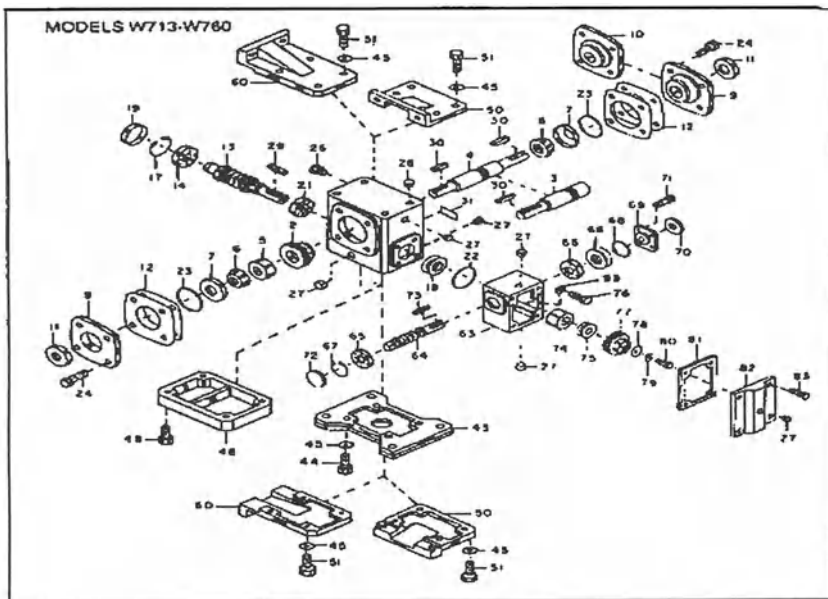


Figure 17.5 The Structure of a Worm and Wormgear Reducer

A simple draft scheme, called assembly draft (KB) is shown in Figure 17.6. By drafting, we can determine the main structure of the reducer and the proper view layout for the final assembly drawing. Further design will be based on this draft, such as the shaft design, determination of the bearing position and type, and finding the load acting position which will be used to test the fit (or misfit) between the shaft, bearings and keys tentative design and the requirements and constraints.

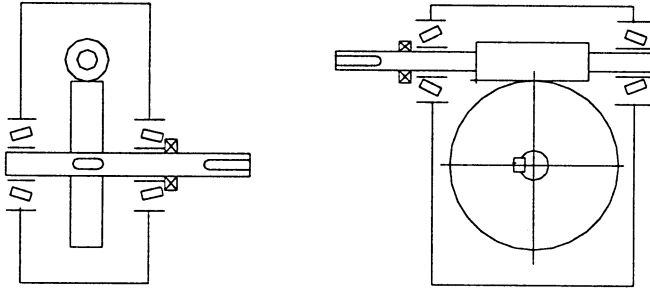


Figure 17.6 The Final Design Concept Selected

17.3.1 MOTOR DESIGN

As calculated above, the power required to raise a $F = 150lb.$ load, at a rate of $V = 30 fpm$, is $P_o = 0.136(hp)$ (C_{16}). The transmission efficiency is low (C_{20}) for the worm and wormgear concept (due to small worm lead angle, C_{19}). So, a transmission efficiency of 60% is selected (C_{21}). Thus, the input power should be:

$$\text{Input power (DP}_{12}) \quad P_i = \frac{P_o}{\eta} = \frac{0.136}{0.6} = 0.227(hp) \quad (5)$$

In order to maintain a certain capacity reservation (C_{22}), a 1/4 hp (DP_{12}), 1500 rpm (DP_{13}), 3-phase AC motor (DP_{14}) will be specified. The motor is selected from manufacturing catalogs (KB, as shown in the Table 17.2) and get all the detail dimension (DP's).

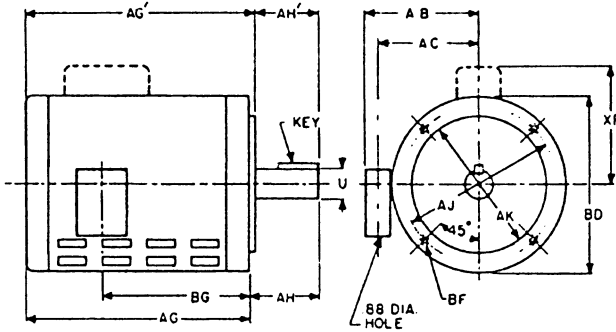
The worm, wormgear and shaft are the main parts of the reducer, which determine the dimensions of other components (from the center of the casing outward). Thus, the designer begins by designing these primary parts, and then considers the other components.

Table 17.2 Dimensions for AC Motors

A.C. MOTORS

NEMA C-FACE

OPEN DRIPPROOF



ALL DIMENSIONS IN INCHES

H.P.	Cat. No.	Item No.	* Mtg. Code	AG	Max. AH	AJ	+0.000 -0.003 AK	BD	AB	BG
1/4	DU-B	66115	II	8.19	2.16	5.875	4.500	6.62	4.50	5.75
	DU-C	66116		7.50				6.50	—	—
	DU-G	66117		7.20				6.52	—	—
	DY-B	66118	II	8.19				6.62	4.50	5.75
	DY-C	66119		7.50				6.50	—	—
	DY-G	66120		7.20				6.52	—	—
1/3	AEU	63597	IA	6.75	1.31	3.750	3.000	4.67	3.41	4.91
	ER-B	66121	II	9.19	2.16	5.875	4.500	6.62	4.50	6.75
	ER-C	66122		8.27				6.50	—	—
	ER-G	66123		8.66				6.52	—	—
	ER5-B	66866	II	9.19				6.62	4.50	6.75
	ER5-C	66867		8.72				6.50	—	—
	ER5-G	66868		9.49				6.52	5.27	5.83
H.P.	Cat. No.	Item No.	Mtg. Code	XP	+0.0000 -0.0005 U	BF Hole Size	Key Depth	Sq.	Lgth	AC
1/4	DU-B	66115	II	—	.6250	3/8-16	.63	3/16	1.38	3.44
	DU-C	66116		—						
	DU-G	66117		—						

17.3.2 THE DESIGN OF THE TRANSMISSION PARTS AND THE OUTLINE OF THEIR RELATIVE POSITION

The transmitting parts of the reducer are worm and wormgear. Figure 17.7 shows the terms used to describe the geometric parameters of the worm and wormgear:

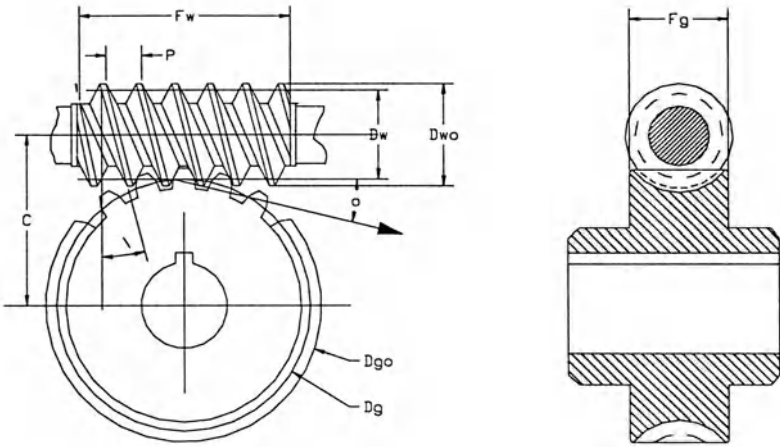


Figure 17.7 The Parameters of the Worm and Wormgear

There are many ways to design the transmission parts (pair). For example, if we choose the pitch P (DP_{15}), thread number N_w (DP_{16}), lead angle λ (DP_{17}) and pressure angle ϕ (DP_{18}) of the worm as an ‘input’; then all other design parameters can be derived. The formulas being used by the designer are taken from the “Machinery Handbooks 22nd revised edition” (KB).

Axial Pitch P is used as the basis for the design standard. Eight standard axial pitches have been established to provide adequate coverage of the pitch range normally required: 0.030, 0.040, 0.050, 0.065, 0.080, 0.100, 0.130, and 0.160 inch. The lead angle λ has a standard 15 series value from 0.5, 1, 1.5, 2, 3, 4, 5, 7, 9, 11,..... to 30 degrees. For our design load (i.e. output reducer power, C_{16}), an axial pitch $P = 0.16$ inch should be strong enough. If we select the worm thread number $N_w = 1$ (since most fine-pitch worms have either one or two threads, KB), lead angle $\lambda = 7^\circ$, pressure angle $\phi = 20^\circ$ (standard pressure angle, KB), then according to the formulas given in the Table 17.3 (KB), we can calculate all other dimensions of the worm and wormgear as follows:

Worm gear teeth number (DP_{19}):
$$N_g = N_w * R = N_w \frac{n_i}{n_o} = 1 * 50 = 50 \quad (6)$$

Worm lead (DP_{20}):
$$l = N_w P = 0.16 \quad (7)$$

Worm pitch diameter (DP_{21}):
$$D_w = \frac{l}{\pi \tan \lambda} = \frac{l}{3.14 * \tan 7^\circ} = 0.415(in) \quad (8)$$

Worm outside diameter (DP₂₃): $D_{wo} = D_w + 2a = 0.415 + 0.101 = 0.516 \text{ (in)}$ (9)

where, $a = 0.3183P \cos \lambda = 0.3183 * 0.16 * 0.9925 = 0.0506 \text{ (in)}$ (DP₂₂) (10)

Wormgear pitch diameter (DP₂₄): $D_g = N_g P / \pi = 50 * 0.16 / 3.14 = 2.548 \text{ (in)}$ (11)

Center distance (DP₂₅): $C_{wg} = 0.5(D_w + D_g) = 1.481 \text{ (in)}$ (12)

Wormgear outside diameter (DP₂₆): $D_{go} = 2C_{wg} - D_w + 2a = 2.648 \text{ (in)}$ (13)

Length of threaded portion (DP₂₇): $F_w = \sqrt{D_{go}^2 - D_g^2} = 0.7236'$ (14)

Wormgear face width (DP₂₉):
 $F_g = 1.125 * \sqrt{(D_{wo} + 2c)^2 - (D_{wo} - 4a)^2} = 0.4941 \text{ (in)}$ (15)

where $c = 0.0637P \cos \lambda + 0.002 = 0.0121$ (DP₂₈) (16)

An alternative way of designing the worm and wormgear is as follows. The designer selects the center distance, C_{wg} , to be approximately 1.5 inch, the pitch $P = 1.60$ inch, and $N_w = 1$. Applying Table 17.4 (KB), the designer finds that there are six possible lead angles: 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, and corresponding worm diameters that will satisfy this lead. The approximate lead angle can be computed from the following formulas given in Table 17.3.

Cotangent of approx. lead angle =

$$\frac{2\pi \times \text{approximate center distance required}}{\text{assumed number of threads} \times \text{axial pitch}} - \text{gearing ratio} \tag{17}$$

So, $\cot \lambda = \frac{2\pi * C_{wg}}{N_w * P} - R = \frac{2 \times 3.1416 \times 1.5}{1 \times 0.16} - 50 = 8.875$ (18)

or, $\lambda = 6.42^\circ$

Table 17.3 Formulas for Proportions of American Standard Fine-pitch Worms and Wormgears (ANSI B6.9-1977)

<p style="text-align: center;">LETTER SYMBOLS</p> <p>$P =$ Circular pitch of wormgear = axial pitch of the worm, P_x, in the central plane</p> <p>$P_x =$ Axial pitch of worm</p> <p>$P_n =$ Normal circular pitch of worm and wormgear = $P_x \cos \lambda = P \cos \psi$</p> <p>$\lambda =$ Lead angle of worm</p> <p>$\psi =$ Helix angle of</p> <p>$N_w =$ Number of threads in worm</p> <p>$N_g =$ Number of teeth in wormgear</p> <p>$R =$ Ratio of gearing = $N_g + N_w$</p>	
---	--

Item	Formula	Item	Formula
WORM DIMENSIONS		WORMGEAR DIMENSIONS **	
Lead	$l = N_w P_x$	Pitch Diameter	$D_g = N_g P + \pi = N_g P_x + \pi$
Pitch Diameter	$D_w = l \div (\pi \tan \lambda)$	Outside Diameter	$D_{go} = 2C_{wg} - D_w + 2a$
Outside Diameter	$D_{wo} = D_w + 2a$	Face Width	$F_g = 1.125 \times \sqrt{(D_{wo} + 2c)^2 - (D_{wo} - 4a)^2}$
Safe Minimum Length of Threaded Portion of Worm *	$F_w = \sqrt{D_{go}^2 - D_g^2}$		
DIMENSIONS FOR BOTH WORM AND WORMGEAR			
Addendum	$a = 0.3183 P_n$	Tooth thickness	$t_n = 0.5 P_n$
Whole Depth	$h_l = 0.7003 P_n + 0.002$	Approximate normal pressure angle	$\phi_n = 20 \text{ degrees}$
Working Depth	$h_k = 0.6366 P_n$	Center distance	$C_{wg} = 0.5(D_w + D_g)$
Clearance	$c = h_l - h_k$		
<p>All dimensions in inches unless otherwise indicated.</p> <p>* This formula allows a sufficient length for fine-pitch worms.</p> <p>** Current practice for fine-pitch worm gearing does not require the use of throated blanks.</p>			

Out of the six possible worms in Table 17.4, the one with 7-degree lead angle is closest to the calculated 6.42° lead angle. Thus, the worm with a pitch diameter $D_w = 0.4148$ inch is selected.

The remaining dimensions of the worm and wormgear may now be determined from the data in Table 17.4, and by using the formulas given in Table 17.3 (KB). The designer finds that the results are almost the same.

Table 17.4 Pitch Diameters of Fine-pitch Worms for American Standard Combinations of Lead and Lead Angle (ANSI B6.9-1977)

Lead in Inches, l	Number of Threads, n	Lead Angle λ in Degrees							
		0.5	1.0	1.5	2.0	3.0	4.0	5.0	7.0
		Pitch Diameter d in Inches							
0.030	1	1.0937	0.5472	0.3647	0.2735
0.040	1	1.4583	0.7297	0.4863	0.3646	0.2429
0.050	1	1.8228	0.9121	0.6079	0.4558	0.3037	0.2276
0.060	2	2.1874	1.0945	0.7295	0.5469	0.3644	0.2731
0.065	1	...	1.1857	0.7903	0.5925	0.3948	0.2959	0.2365	...
0.080	1,2	...	1.4593	0.9726	0.7293	0.4859	0.3641	0.2911	...
0.090	3	...	1.6417	1.0942	0.8204	0.5466	0.4097	0.3274	0.2333
0.100	1,2	...	1.8242	1.2158	0.9116	0.6073	0.4552	0.3638	0.2592
0.120	3,4	...	2.1890	1.4590	1.0939	0.7288	0.5462	0.4366	0.3111
0.130	1,2	1.5805	1.1851	0.7896	0.5917	0.4730	0.3370
0.150	3,5	1.8237	1.3674	0.9110	0.6828	0.5457	0.3889
0.160	1,2,4	1.9453	1.4585	0.9718	0.7283	0.5821	0.4148
0.180	6	2.1884	1.6408	1.0932	0.8193	0.6549	0.4667
0.195	3	1.7776	1.1843	0.8876	0.7095	0.5055
0.200	2,4,5	1.8232	1.2147	0.9104	0.7276	0.5185
0.210	7	1.9143	1.2754	0.9559	0.7640	0.5444
0.240	3,6,8	2.1878	1.4576	1.0924	0.8732	0.6222

17.3.3 TESTING THE CURRENT DESIGN AGAINST THE WORMGEAR LOAD AND STRENGTH CONSTRAINTS

The teeth of the wormgear are always weaker than the gear teeth (KB). Therefore, the bending stress and wear strength of the wormgear should be much less than the allowable static stress of the wormgear material (C_{23}). For future design (of the shaft,

bearings and keys), and strength checking, we need to know the load forces acted on the worm, wormgear and shaft as shown in Figure 17.8.

The normal force, F_n (C_{24}), acts perpendicularly to the tooth surface, and is broken into three components: the tangent force, F_t (C_{25}), the radial or separating force, F_r (C_{26}), and the thrust force F_{thrust} (C_{27}). The designer notices that the forces would not exist unless certain power is transmitted through the worm and wormgear (KB). In other word, the load forces are the function of the transmitted power, e.g. $F = f(P)$. By studying the literature (e.g., [1]), the designer calculates the forces as follows (KB):

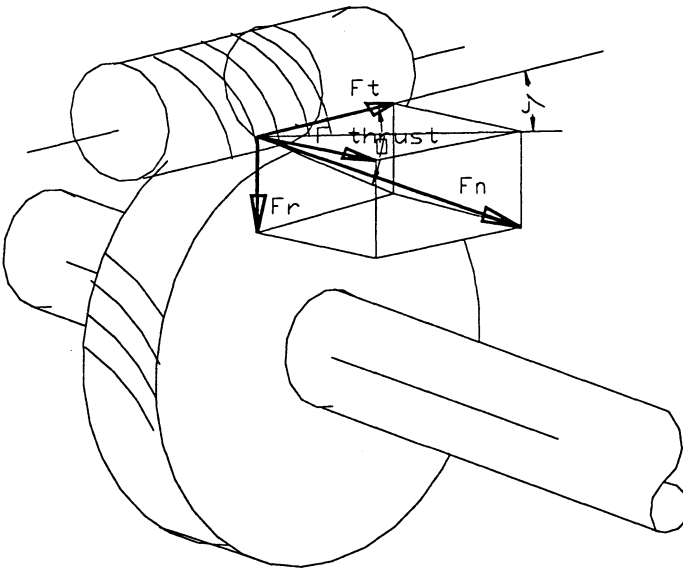


Figure 17.8 The Forces Acted on the Wormgear

$$F_t = \frac{T}{0.5D_g} = \frac{P_o * 63000}{N_o * 0.5D_g} = \frac{0.136 * 63000}{3057 * 0.5 * 2.548} = 220(lb) \tag{19}$$

$$F_n = \frac{F_t}{\cos \phi \sin \lambda} = \frac{220}{\cos 20^\circ \sin 7^\circ} = 1921(lb) \tag{20}$$

$$F_r = F_n \sin \phi = 1921 * 0.342 = 657(lb) \tag{21}$$

$$F_{thrust} = F_n \cos \phi \cos \lambda = 1921 * 0.9397 * 0.9925 = 1791(lb) \tag{22}$$

The designer applies the simplified Lewis equation (KB) to the wormgear in determining its strength. First, the bending strength (C_{32}) is calculated. From Lewis equation:

$$S = \frac{F_d P}{Y F_g} \quad \text{where,} \quad (23)$$

$$\text{Dynamic bending load (} C_{31}\text{):} \quad F_d = \left(\frac{1200 + V_{pg}}{1200} \right) \times F_t \quad (24)$$

where, V_{pg} (C_{30}) is the wormgear pitch line velocity which is given by:

$$V_{pg} = n_o 2\pi D_g / (2 \times 12) = \frac{30 \times 2 \times 3.14 \times 2.548}{2 \times 12} \approx 20 \quad (\text{feet per second}) \quad (25)$$

S is the static stress (C_{28})

Y is Lewis form factor (C_{29}). For $\phi = 20^\circ$, $Y = 0.392$.

The designer already calculated the normal circular pitch $P=0.16$ (DP_{15}), and the width of the gear $F_g=0.4941 \approx 0.5$ (DP_{29}). Thus,

$$F_d = \left(\frac{1200 + 20}{1200} \right) \times 220 = 224(\text{lb.}) \quad (26)$$

Therefore, the bending stress on the wormgear tooth can be calculated as:

$$S = \frac{F_d P}{Y F_g} = \frac{224 \times 0.16}{0.392 \times 0.5} = 179.2(\text{psi}) \quad (27)$$

This value is much less than the allowable static stress of a phosphor bronze, $S_o = 12,000$ psi. Therefore, the wormgear material is chosen to be phosphor bronze (DP_{30}).

The designer needs also to check the wear strength of the wormgear. Due to space limitations, we skip these calculations (for details, refer to [2]).

If the wear strength is not satisfactory with the allowable strength requirement, we must either change the material or change the normal circular pitch P .

With this information, we can either purchase the worm and wormgear or manufacture them.

17.3.4 INITIAL DESIGN OF THE CASING (BOX)

As shown in Figure 17.9, certain distances must be maintained between the inner wall

and the outside diameters or end faces of the transmitting parts. Δ_1 , the distance between the outside diameter of the wormgear and the inner wall (DP_{31}), is usually equal to or larger than δ , the thickness of the wall (C_{30}). Δ_2 , the distance between the end face of the wormgear and the inside wall (DP_{32}), is usually equal or larger than 1.2δ (C_{31}).

Table 17.5 The Main Structural Dimensions of the Cast Iron Casing (unit: millimeter)

items	symbols	dimensions
thickness of casing wall distance	δ	$0.04a + 3 \geq 8$ a: center distance between worm and wormgear
thickness of cover wall	δ_1	$\approx \delta$ (for worm on top), $= 0.85 \delta$ (for worm on bottom)
thickness of cover edge	b_1	$1.5 \delta_1$
thickness of casing edge	b	1.5δ
thickness of casing bottom edge	b_2	2.5δ
diameter of installation screw	d_f	$0.036a + 12$
number of installation screws	n	4
diameter of cover screw	d_2	$0.5 \sim 0.6 d_f$
space of cover screw	l	150 ~ 200
diameter of bearing cover screw	d_3	$0.4 \sim 0.5 d_f$
diameter of location pin	d_p	$0.7 \sim 0.8 d_2$
height of bearing protrude	h	based on the bearing outer ring; must be easy for wrench operation
distance for gear outer diameter to casing inner wall	Δ_1	$> 1.2 \delta$
distance from gear end face to casing inner wall	Δ_2	$> \delta$
rib thickness	m	$\approx 0.85 \delta$
outer diameter of bearing cover	D_2	$= D + 5 d_3$ D: bearing outer diameter $= D + 5 d_3 + (15 \sim 20)$ (for invalid bearing cover)

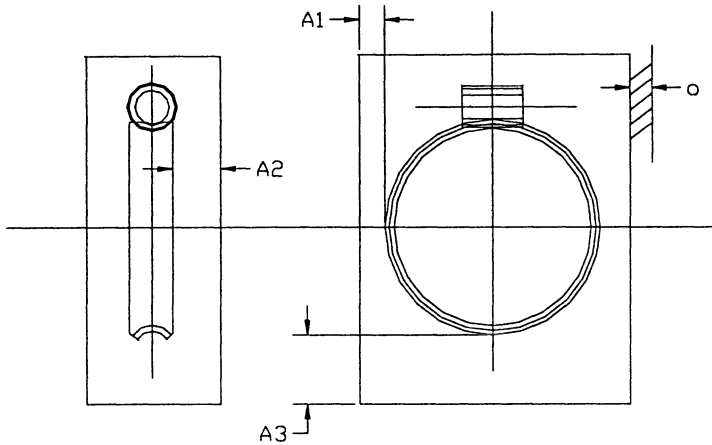


Figure 17.9 The Distances Between the Transmission Parts and Casing Inner Wall

Most of the structural dimensions of the casing are related to the dimensions of the transmitting parts (KB). The designer notices that the distance between the wormgear end face and the inner wall, Δ_2 , is much farther than Δ_1 because of considerations related to the dimensions of the bearings on the worm shaft (KB).

For the position of the casing bottom, we must consider the lubrication and cooling for the mechanism. Certain space volume is needed to keep oil and allow the dirty substance to precipitate (C_{32}). Thus, the distance between the outside diameter of the wormgear and the inner wall of the bottom, Δ_3 (DP_{33}) should be larger than $8\sim 12 P$, and not less than 1.25 in. (C_{33}).

After the determination of the positions of the inner walls, the designer determines the maximum overall dimensions of the casing according to the thickness of the wall and the size of rib. For example, by the position and maximum size of the connecting parts outside the casing (such as the couplings), the designer can determine the length which the shaft extends out of the casing.

Table 17.5 provides the experienced reference values (Dp 's) for the casing design of a worm-gear reducer (KB). The values were determined mainly by considerations of strength and rigidity (C 's), as well as compactness and machinability (C 's). Notice that the thickness of the wall (δ) remains (at this stage) undetermined.

17.3.5 THE DESIGN OF THE WORMGEAR SHAFT SET

After designing the transmission parts and the casing, the diameter of the shaft is *initially* calculated according to the maximum allowable torsion strength (C_{34}). A maximum torsion strength is obtained, if the transmission power equals the input power (C_{35}). Therefore, the designer uses the following formula (KB):

$$\text{Diameter of the shaft (DP}_{34}\text{): } d = \sqrt[3]{\frac{80P_t}{n_o}} \text{ (in)} \quad d = \sqrt[3]{\frac{80 * 0.25}{30}} = 0.8735 \text{ (in)} \quad (28)$$

The designer decides to chose 1 inch as the diameter of the shaft (DP₃₄).

Since a large thrust force (F_{thrust}) is created by the worm and wormgear, a tapered roller bearing must be chosen (DP₃₅), and the material used for the shaft should be commercial steel (DP₃₆). To simplify the shape of the shaft (C₃₆), a one inch diameter will go through all the length of the shaft (DP₃₇). To keep the distances between the gear and the bearings (C₃₇), two sleeves (DP₃₈) will be used. For the shaft to be axially fixed only by the press fit of the bearing assembly (C₃₈), and for the thrust force to be suffered by the end covers through sleeves and bearings (C₃₉), the axial position and clearance in the bearings will be adjusted by the end covers against the bearing's outer rings (DP₃₉, see Figure 17.10).

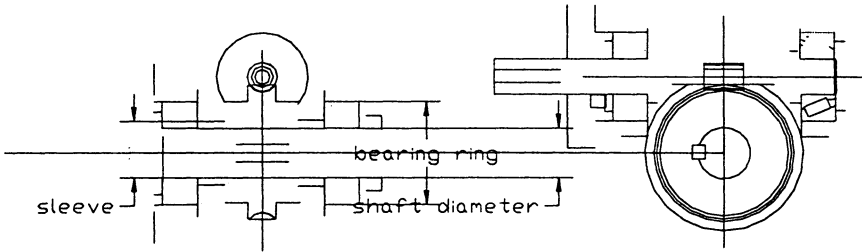


Figure 17.10 The Radial Dimension of the Shaft

To determine the axial dimension of the shaft, we need to consider the axial size (width) of wormgear, sleeve and bearing. The width of the wormgear is only 0.5 in. (DP₂₉). For stable and rigid assembly (C₄₀), and high strength of the key (C₄₁), the fit length of the shaft and gear must be larger than the diameter of the shaft (C₄₂). Therefore, the designer chooses 1.5 inch as the axial length of the wormgear (DP₃₀). From the bearing catalog (KB), he/she finds that the width of the bearing (DP₃₁), and the width of the outer ring of the bearing (DP₃₂) which correspond to a 1 inch bore are 0.875 inch and 2.5625 in., respectively. For stable and rigid assembly (C₄₀), a safety flange couplings are used (DP₃₃), and a seal unit will be needed at the extended end of the shaft. For the 1.5 inch axial length of the shaft, the width of the seal unit is 0.437 inch (DP₃₄), as selected from the catalog (KB). The assembly draft as Figure 17.11.

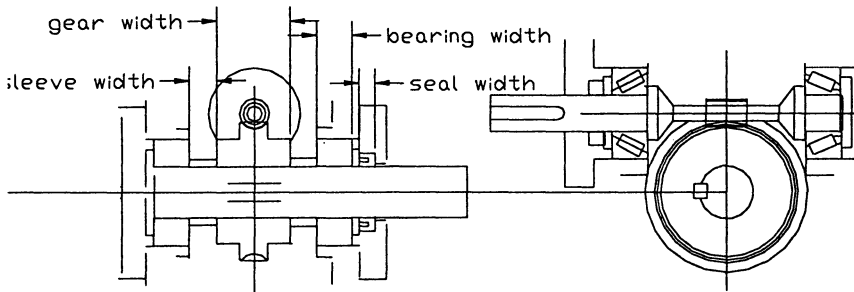


Figure 17.11 The Initial Drawing of the Assembly Draft

17.3.6 CALCULATION AND CHECK OF THE SHAFT SET PARTS

In this step, the designer checks the strength of the shaft set parts, such as shaft, key and bearings. If some of them are not satisfactory with the strength, rigidity, or durability constraints, some modification or even redesign of the shaft set parts will be needed.

Shaft Load Calculation

The strength of the shaft should be less than the maximum allowable shearing stress (C_{42}). Therefore, the designer has to know the bending (C_{43}) and twisting load (C_{44}) on the shaft. To calculate the loads, the designer uses the magnitude, direction and the acting point of the external forces F_t , F_r , F_l , F_r , F_{thrust} , and the torque T (C_{45}). By using the knowledge of Mechanics (KB), the designer calculates all the support forces D_{max} (C_{46}), and B_{max} (C_{47}), and the maximum bending moment, M_{max} (C_{48}), and torsion moment T_{max} (C_{49}). Then, he/she uses the combined bending and torsion moments to test the previously calculated diameter of the shaft.

To calculate the support forces, the designer draws a straight line A-B-C-D, representing the shaft as shown in Figure 17.12:

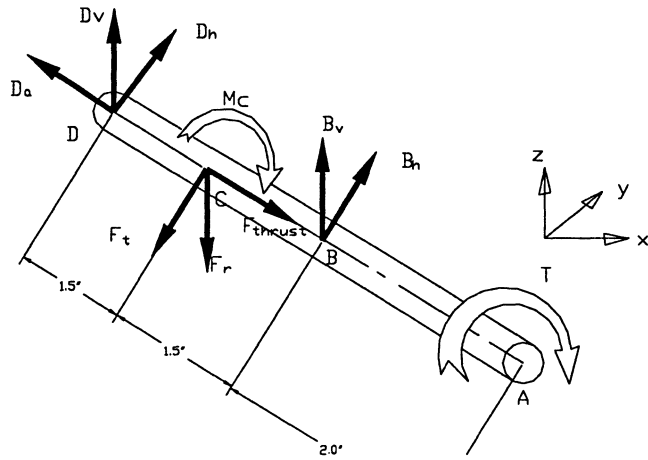


Figure 17.12 Schematic Force System on the Shaft

The support forces at points B and D (bearings) are:

$$B_{\max} = \sqrt{B_v^2 + B_h^2} \tag{29}$$

$$D_{\max} = \sqrt{D_v^2 + D_h^2} \tag{30}$$

and the maximum torque is :

$$T_{\max} = T_{AC} \text{ (the torque between the points A and C), where}$$

$$C_{50}: T_{AC} = 0.5D_g F_t = 0.5 * 2.548 * 220 = 280.28(\text{lb} \cdot \text{in}) \tag{31}$$

On the vertical plane B_v (C_{51}) and D_v (C_{52}) satisfy:

$$B_v + D_v = F_r \tag{32}$$

and for point B, M_c (C_{53}) satisfies:

$$-1.5F_r + 3D_v + M_c = 0; \quad \text{and} \tag{33}$$

$$M_c = 0.5D_g F_{thrust} = 0.5 * 2.548 * 1791 = 2281.7(\text{lb} \cdot \text{in}) \tag{34}$$

Thus, $D_v = -310(\text{lb})$ (negative means that the force is in the opposite direction of the drawing), and $B_v = 967(\text{lb})$.

On the horizontal plane, B_h (C_{54}) and D_h (C_{55}) satisfy:

$$B_h + D_h = F_t ; \quad \text{and} \quad (35)$$

$$B_h = D_h = F_t / 2 = 110(lb) \quad (36)$$

On the axial direction, D_a (C_{56}) satisfies:

$$D_a = F_{thrust} = 1791(lb) \quad (37)$$

The designer proceeds to draw the shear and bending moment diagrams (KB) for the horizontal plane and vertical plane as shown in Figure 17.13, and calculates the maximum combined bending moment at section C:

$$M_{max} = \sqrt{1450^2 + 165^2} = 1,459(lb \cdot in)$$

The support forces at points B and D (bearings) are:

$$B_{max} = \sqrt{B_v^2 + B_h^2} = \sqrt{967^2 + 110^2} = 973(lb)$$

$$D_{max} = \sqrt{D_v^2 + D_h^2} = \sqrt{(-310)^2 + 110^2} = 328.9(lb)$$

Notice that when the worm rotates in the opposite direction, the forces F_{thrust} , F_t , D_a , B_v , and D_v will be changed directionally or quantitatively.

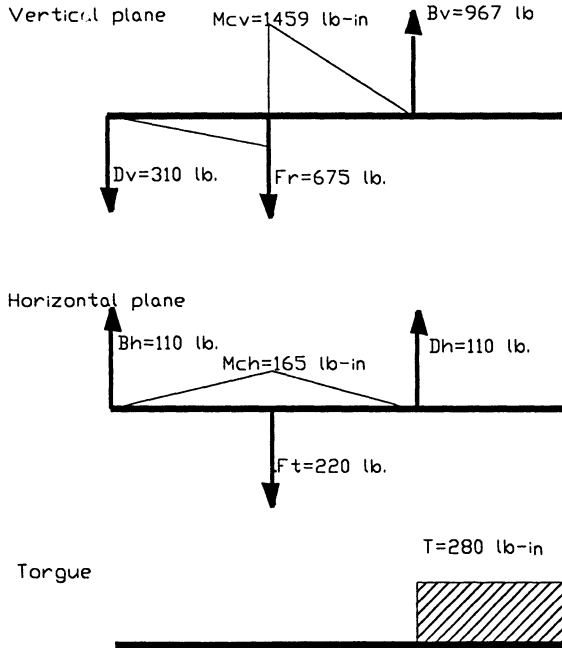


Figure 17.13 Load, Shear, and Moment Diagrams for the Shaft

Shaft Strength Testing

The shaft we are designing is subjected to combined torsion and bending. By using the previously calculated values and the Machinery's Handbook 22nd revised edition (KB), the diameter of the shaft should satisfy (C₅₇):

$$d \geq 3 \sqrt{\frac{5.1}{P_{ij}} (K_m M_{\max})^2 + (K_t T_{\max})^2} \quad (38)$$

K_m (C₅₈) and K_t (C₅₉) are combined shock and fatigue factors for bending and torsion, respectively. The Machinery's Handbook (KB) shows that $K_m = K_t = 1$. $M = M_{\max} = 1459$ is the maximum bending moment, and $T = T_{\max} = 280 \text{ lb} \cdot \text{in}$ is the maximum torque (see the above calculations). P_{ij} is the maximum allowable shearing stress under combined condition (C₆₀). Since the material used for the shaft set is commercial steel (DP₃₆ above), and the key has high strength (C₄₁), P_{ij} is found (in the designer KB) to be 8,000 psi.

Thus,

$$d = 3 \sqrt{\frac{5.1}{6,000} (1459^2 + 280^2)^{1/2}} = 0.98 \leq 1(\text{in})$$

This number is less than the previously calculated value of 1 inch. Thus, the strength of the shaft is good enough. If there is a misfit between Constraint C₅₇, the designer needs to redesign the shaft by either increasing its diameter (DP₃₄) or changing its shape (DP₃₇).

Testing the Strength of the Key

The designer selects the key size (DP₄₀) according to the diameter of the shaft. For the 1" diameter shaft, the parallel key should have (according to the Machinery's Handbook, KB) a section dimension of 1/4" x 1/4" x 1" (W x H x L). The shear strength and compression stress of the key (see Figure 17.14) must be less than the allowable shear strength and compression stress, respectively (C₆₁). To test whether the key satisfies Constraint C₆₁, the shear stress (C₆₂) and compression stress (C₆₃) on the key should be calculated, and its material (DP₄₁) should be specified:

$$\text{Shear stress (C}_{62}\text{): } S_s = \frac{F_s}{A_s} = \frac{2T}{dWL} = \frac{2 * 280}{1 * 1/4 * 1} = 2240(\text{psi}) \quad (39)$$

$$\text{Compression stress (C}_{63}\text{): } S_c = \frac{F_c}{A_c} = \frac{4T}{dWL} = \frac{4 * 280}{1 * 1/4 * 1} = 4480(\text{psi}) \quad (40)$$

Material of the key (DP₄₁): ASTM 40. The designer finds in the Machinery’s Handbook (KB) that the calculated values of S_s and S_c are much less than the allowable stress values for shearing and compression. Therefore, the key selection is safe (in the evolutionary design model jargon, ‘there is a fit between the strength and stress constraints and the key design’).

If the constraints are not satisfied by the key, (either by shearing or compression), the designer has either to change the material, or change the shaft diameter (the key size is associated with the diameter of the shaft).

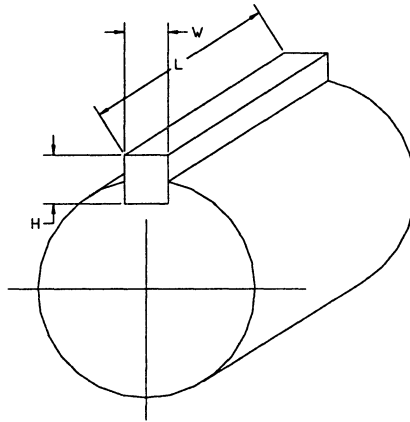


Figure 17.14 The Shearing and Compression Stress of the Key

Testing the Rating Life of The Bearings

To test if our tentative design satisfies the duration constraint (C₁₇, above), the rating life of the selected bearing must be greater than the specified duration (C₆₄). The designer uses the following equation (given in the manufacture’s catalog, KB):

$$C_{67}: \quad L_{10} = \frac{10^6}{60n_b} \left(\frac{C}{P_e}\right)^b \tag{41}$$

L_{10} is the rating life of the bearing in hours of life, resulting in 10 % failure; C is the basic load rating in pounds (C₆₅); P_e is the equivalent load in pounds (C₆₆); $b = 3.0$ for ball bearings and $10/3$ for roller bearings (KB); and n_b is the rotational speed in revolutions per minute (C₆₇).

To calculate the equivalent load, the designer uses apply the following formula:

$$C_{71}: \quad P_e = XvF_r + YF_a \tag{42}$$

where F_r = radial load (C₆₈), lb.; F_a = thrust load (C₆₉), lb.; v = rotation factor = $1 \sim 1.2$ (C₇₀); X = a radial load factor (retrieved from the handbook, KB); and $Y = a$

thrust load factor (retrieved from the handbook, KB). Figure 17.15 shows the load forces on a pair of tapered roller bearings:

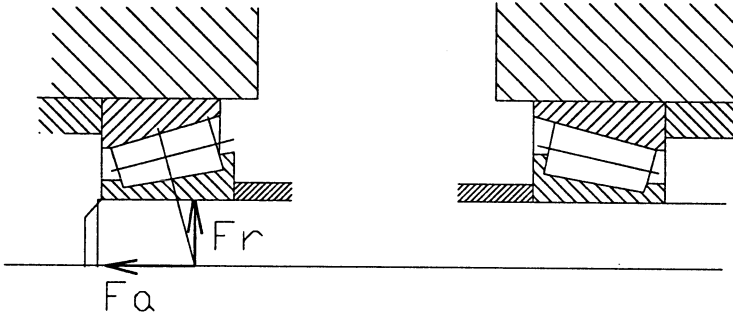


Figure 17.15 The Load Forces on A Pair of Tapered Roller Bearings

From the engineering handbook or catalog (KB), the designer finds that for the bearings that were selected (1 inch bore diameter, single row tapered roller bearing):

$\nu = 1$; $X = 0.4$; $Y = 1.17$; $C_o =$ basic static load rating = 4,650 lb; and $C =$ basic dynamic load rating = 6,060 lb.

The designer also knows (KB) that:

$$C_{72}: \quad F_r = B_{\max} = 973 \text{ lb.} \quad (43)$$

$$C_{73}: \quad F_a = D_a = 1791 \text{ lb.} \quad (44)$$

$$C_{74}: \quad n_b = n_o = 30.57 \text{ rpm} \quad (45)$$

Thus: $P_e = 0.4 \times 1 \times 973 + 1.17 \times 1791 = 2484.67 \text{ lb.}$; and $P_{10} = 10,841.4 \text{ hours.}$

Therefore, there is a fit between the current design and the requirement for 5 years product life, and 90% reliability.

17.3.7 STRENGTH AND WEAR-RESISTANCE CONSTRAINTS

The strength and wear-resistance of the shaft should be within pre-specified limitations. Since the basic functional requirement of the hoist is to lift light objects (C_2) or opening a garage door (C_3), and the hoist includes a wormgear reducer (DP_6), the basic functional requirement of the wormgear is to transmit power (C_{75}). Therefore, the strength and wear-resistance of the shaft are good within the pre-specified limitations.

If the shaft requires precise transmission (such as in a lathe spindle operation), the designer needs to check the bending and torsion deflections to make sure they are within the pre-specified limitations.

17.3.8 DETAILED DESIGN OF THE CASING

The casing is the part which support and fix the shaft set, ensure the rotational precision, maintain the condition of lubrication and seal. We design the casing based on the shaft set.

To ensure the rigidity of the casing (C_{76}), the reasonable thickness of the wall is very important (C_{77} , see Figure 17.16). Thus, the designer uses (based on previous experience, KB), the following formula:

$\delta = 2\sqrt[4]{0.1T} \geq 6 \sim 8(mm)$ (the unit of T here is Newton-meter) to get the 5/16 inch thickness of the wall (DP_{42}).

The casting iron (DP_{43}) is usually used for casing material. For large casing (C_{78}), the designer needs to add some reinforce plate (strengthening rib, DP_{44}). Since the linear speed of the wormgear reducer is below 40 feet per second, oil is usually used as the lubrication and cooling medium (DP_{45}). The casing must have enough space to store the oil (C_{79}). The volume of the oil which is needed is counted based on the power transmitted (0.35 ~0.7 liter oil per kilowatt). Therefore, at least 50 to 100 milliliters oil is needed to keep in the casing or about 3 to 6 cubic inches (DP_{46}). The minimum soaking depth which the gear tooth is below the oil surface (DP_{47}), is a full length of the tooth and no less than 0.5 inch (C_{80}). To keep from over disturbing the oil (C_{81}), a 1 to 2 inches distance from the tooth top to the oil bottom must be kept (DP_{48}).

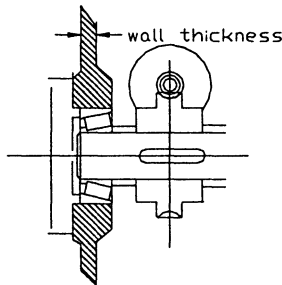


Figure 17.16 The Increased Thickness is Employed to Strengthen the Local Rigidity

17.3.9 ACCESSORIES DESIGN

There are many accessories which are related to the casing that can to be considered,

such as peep hole, sight glass, oil drain plug (DP's) shown in Figure 17.17. Most of these accessories are standard parts the designer can chose from manufacturing catalogs (KB).

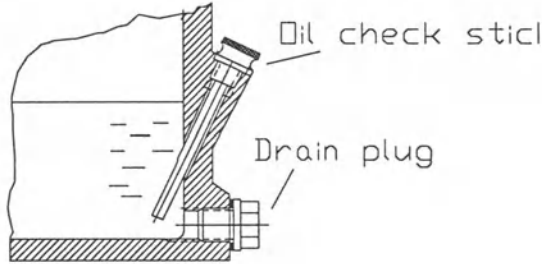


Figure 17.17 Accessories Design

17.3.10 CASING HEAT BALANCE CONSTRAINTS

The worm and wormgear creates more heat than other transmission mechanisms. Heat balance is usually needed to prevent the overheat (C_{82}). If the casing does not meet the requirement, the size or surface of the casing should be increased, or some cooling devices, such as fan and radiator, should be added to the casing (new DP's).

We will not pursue the development of this gear box any further, except to note that eventually the gear box was designed in detail according to the evolutionary model and manufactured in the ADMS Laboratory at Boston University. The unfinished assembly working drawing of the worm and wormgear reducer is shown in Figure 17.18. The main resulting dependencies (up to this stage) between constraints and design parameters is shown in Figure 17.19.

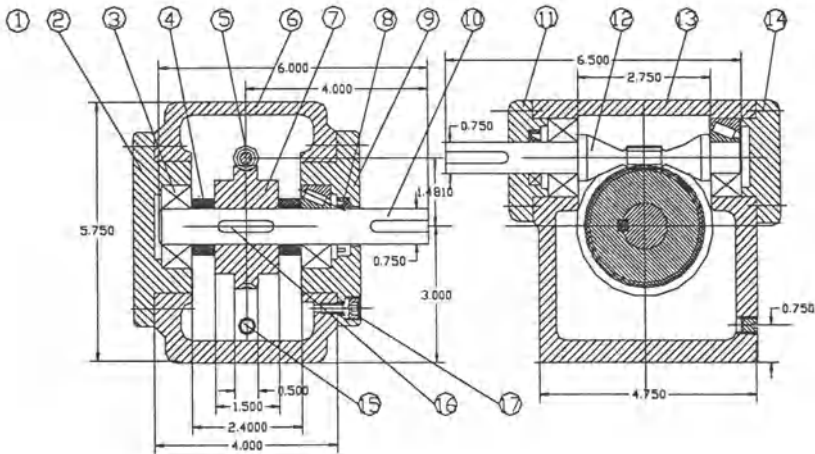
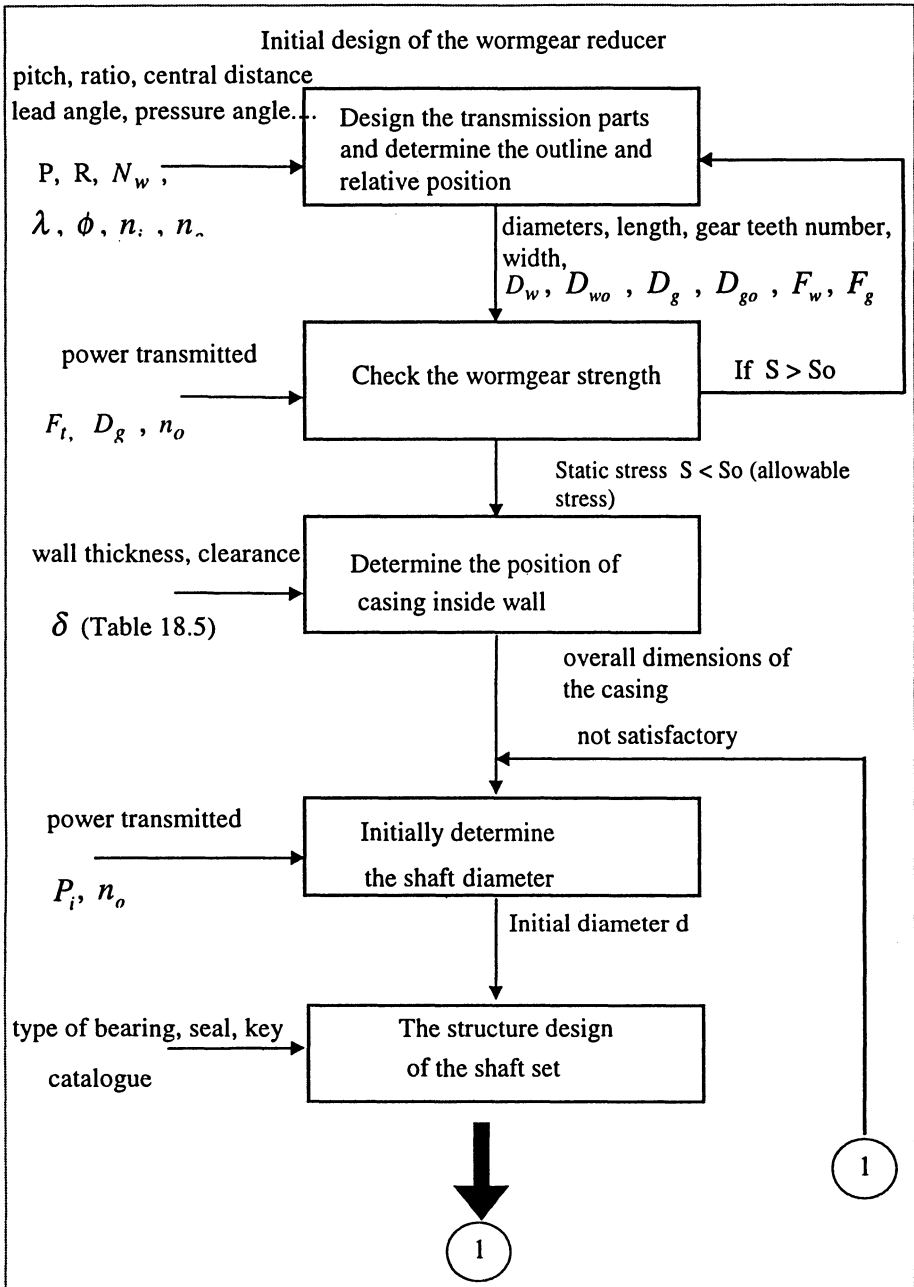


Figure 17.18 Assembly Working Drawing of the Worm and Wormgear Reducer



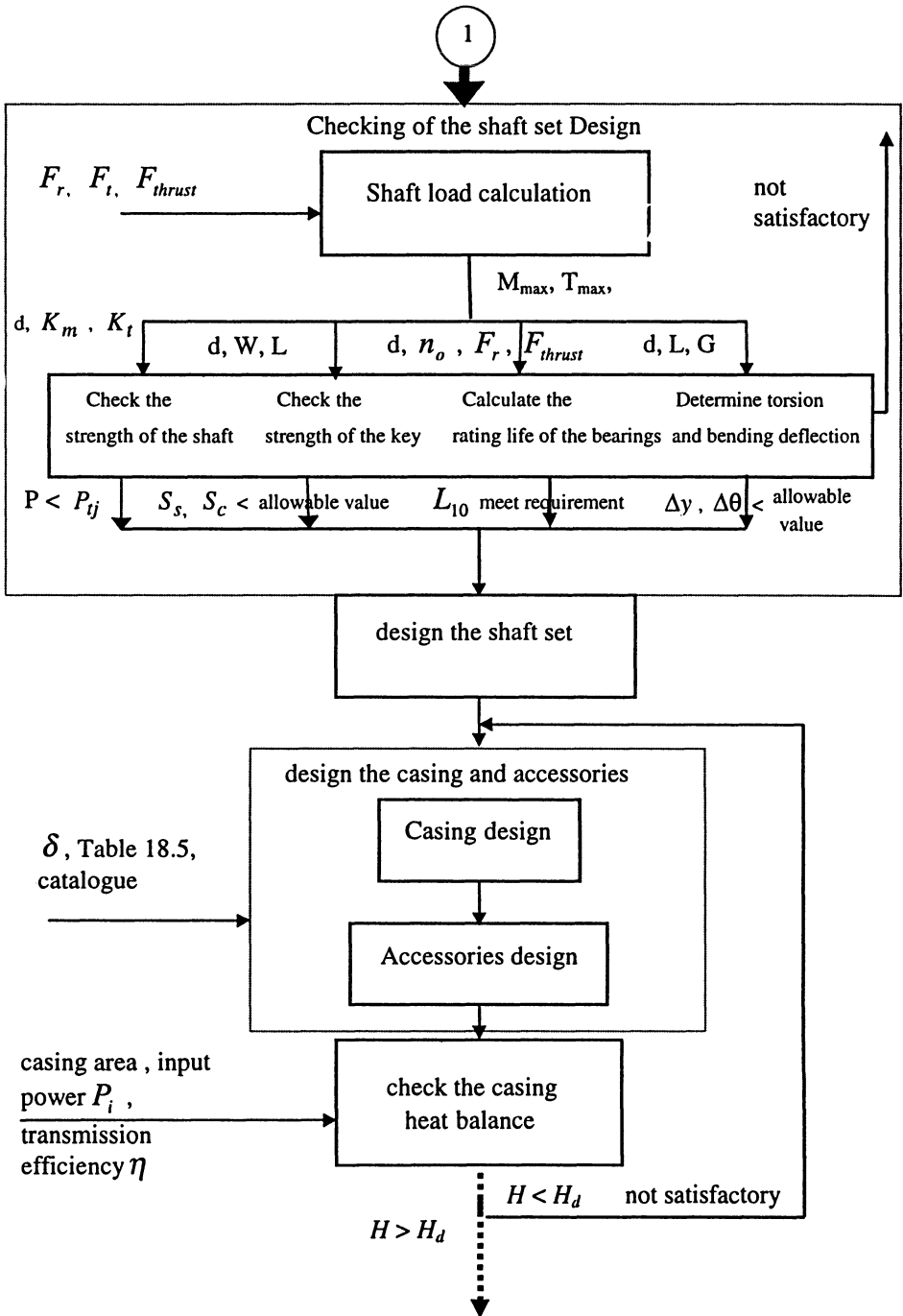


Figure 17.19 A Snapshot Of the Constraints and Design Parameter Graph

17.4 DISCUSSION

The theory of evolutionary design process is a paradigm which, in the final analysis, is founded on a simple unifying notion: that a design process is to be viewed as an alternation between ‘specifications’ (requirements and constraints) and ‘design parameters’. In the evolutionary model, the design process begins with some *goal* constraints. It becomes the designer task to make these constraints matter of *fact*. In the case of the Gear Box design, the initial requirements were to design “a mass production device, which can be used for lifting light objects or opening a garage door in a family or small warehouse.” These requirements were assimilated into the Gear Box design in the form of three constraints (C_1), (C_2) and (C_3).

The validity of constraints (*qualitative* design requirements as well as *closed-form equations*) as claimed by the designer is determined by the kind of evidence invoked in support of the constraints. The evidence in turn, involves relevant *knowledge* of the design domain; *tools* and *techniques* of verification (finite-elements or finite-differences); and the *causal history* of the individual constraints and design parameters as illuminated by the constraints and design parameters dependency graph (see Figure 17.3). Consider the following examples:

- The constraint C_{11} (“to insure a long work life for the rope”) is considered validated only if constraint C_{12} (“the overbending of the rope is avoided”) is validated. This in turn, is considered validated only if constraint C_{13} (“the minimum drum diameter should be large enough”) is validated. According to the engineering recommendation value (knowledge of the design domain), for the minimum drum diameter to be large enough, a 15 to 20 times the rope diameter should be chosen. Therefore, the designer chooses the drum diameter as $d_{dr} = 3.75$ inches (DP_9). The *tool* used for support of constraint C_{13} is simply to check if the selected drum diameter is indeed 15 to 20 times the rope diameter.
- The constraint C_{14} ($n_o = 12V/\pi d_{dr}$) is considered validated only when the constraints $n_o = 30.57$ rpm and $V = 30$ feet per hour, and the design parameter $d_{dr} = 3.75$ were specified. Notice that, for completeness, we should have distinguished between the constraint $n_o = 30.57$ rpm, and the constraint $n_o = 12V/\pi d_{dr}$. However, due to space limitation (and readability), we omitted this notation from the Gear Box thought experiment.
- The validity of the constraint “select an efficient reducer” is determined by a concept selection process (the *verification tool*), which is used to measure the degree of efficiency of each alternative with respect to well-defined criteria. This in turn, is considered validated only if DP_{11} (“the reducer is a worm and wormgear type”) is validated (or selected).

The constraints and design parameters have dependencies among them. Dependencies between the constraints and parameters (see Figure 17.3) are

represented by *production rules*, which describe the logical relationships between the *antecedent* and *consequent* parts (constraints or design parameters). The consequent part is considered validated only if the antecedent part is validated. The production rules are specified in the first-order calculus. For example, the validity (or value) of constraint C_{25} : $F_t = \frac{P_o * 63000}{N_o * 0.5D_g}$ (which can be formulated in first-order calculus) is

determined by the logical relationship: $(F_t = 220(lb) \wedge P_o = 0.136 \wedge n_o = 30.57 \wedge D_g = 2.548)$. In this case, the *specification part* is updated by replacing constraint C_{25} by its *antecedents* (C_{25} is called *consequent*).

As another example, the validity of constraint C_{61} : “the shear strength and compression stress of the key must be less than the allowable shear strength and compression stress, respectively “ is determined by the logical relationship: (“the shear stress should be calculated” \wedge “the compression stress should be calculated” \wedge “the key’s material should be specified” \wedge “the shear stress must be less than the allowable shear stress of the material” \wedge “the compression stress must be less than the allowable compression stress of the material”). In this case, the specification part is updated by replacing constraint C_{61} by its *antecedents* (C_{61} is called *consequent*). Note that the constraint “the key’s material should be specified” derives a new design parameter DP_{41} : “the material of the key is ASTM 40.” In this case, the *design part* is updated by adding the new design parameter DP_{41} .

The evolutionary nature of the Gear Box design process (with respect to the evolutionary schema described in Chapter 6) is quite explicit. In the Gear Box design process, the objects that evolve are the design/constraints complexes; the *direction* of evolution is towards the attainment of satisfied constraints; and the *mechanism* of evolution is the attempt to verify the validity of existing constraints, and as a consequence the introduction of new constraints and design parameters. Table 17.6 relates the design theory and the Gear Box design process:

Table 17.6 Comparison Between the Design Evolutionary Theory and Gear Box Design Process

Evolutionary design model	Gear Box design process
Design + Specifications	Set of user-specified requirements, equations, design attributes and design parameters (dimensions)
Verifications of the specifications versus the design	Attempt to verify the validity of qualitative constraints and equations
Identifications of cause of discrepancy	Identifications of constraints that cannot be validated
Misfit elimination	Generation of new design parameters or new constraints

A most distinct feature of the design evolutionary model (which implicitly assumed) is that the design process is *constantly subject to revision*. For example, in

the Gear Box design process, if the shear strength and compression stress constraints are not satisfied by the key, the designer has either to change the key material (DP_{41}), or change the shaft diameter DP_{34} (the key size is associated with the diameter of the shaft). In that case, all constraints that dependent on DP_{41} and/or DP_{34} will have to be *revised* in the light of this new design state. The evolutionary design process is, thus, *non-monotonic* in nature. Another important feature is that the evolutionary design process pattern is not unique. For example, in designing the worm and wormgear, the designer can either initially specify the axial pitch P , lead angle λ , worm thread number $N_w = 1$, and pressure angle ϕ ; or selecting the center distance, C_{wg} , axial pitch P , $N_w = 1$, and lead angle λ .

Having demonstrated the evolutionary theory of design, we now show¹ how the methodology of the Gear Box design can be given a formal description according to the model presented in Chapter 6. A formal description will be useful for automatically implementing the evolutionary design methodology.

As shown in Chapter 6, a *Type-2 Design Process* is denoted as a tuple $DP = \langle L, Q, P, T, S_0, F \rangle$, where L denotes the *design description*, which includes both structural and specification attributes; Q is a finite set of *process states* (conjunction of structural and functional attributes); P is a finite set of *production rules* -- retrieved from the designer's knowledge body -- which are suggested as a means of changing the current process state to a modified state; T is an operator, which is simultaneously used for decomposition of specifications and matching the specifications with partial solutions; S_0 is the initial process state; and F is the set of terminal states. A process is a series of *transformations*. The series exhibits a precedence relationship among its transformed *process states*.

The Gear Box design process can be shown to be an instance of this general evolutionary process model with the definitions given as follows:

17.4.1 DESIGN DESCRIPTION (L)

L is a first-order logic. In the Gear Box example, the finite set of structural attributes in L includes (1) *part types* (group of parts which are similar but are of different sizes) such as 'drum', 'motor' and 'wormgear'; (2) *qualitative attributes* given to part types such as 'pitch', 'lead angle', 'pressure angle' and 'gear teeth number'; and (3) *attribute values*, which are real numbers assigned to qualitative attributes (e.g. pitch angle = 0.16). Although the *attribute values* are real numbers and, thus, are continuous, finiteness of the domain is a reasonable assumption here since the values of the dimensions can be quantized at the level of the desired tolerance.

The finite collection of functional (specification) attributes in L includes *qualitative specifications* and *quantitative specifications (closed-form constraints)*. In the Gear Box design, qualitative specifications include: {"a mass production device, which can be used for lifting light objects or opening a garage door in a

¹ For complete description of the formal model refer to [2]

family or small warehouse”, “to insure a long work life for the rope”, “the shear strength and compression stress of the key must be less than the allowable shear strength and compression stress, respectively “}. *Closed-form constraints* include:

$\{ F_t = \frac{P_o * 63000}{N_o * 0.5D_g}, B_{\max} = \sqrt{B_v^2 + B_h^2}, D_w = 0.415(in) \}$. We can also have: (“to insure a long work life for the rope” $\wedge D_w = 0.415(in)$).

The artifact part (or part of it) may satisfy the specification part. For example, the bending stress on the wormgear tooth was calculated to be:

$S = \frac{F_d P}{Y F_g} = \frac{224 \times 0.16}{0.392 \times 0.5} = 179.2(psi)$. This value is much less than the allowable

static stress of a phosphor bronze, $S_o = 12,000$ psi. Therefore, the wormgear material (a structural attribute) satisfies the user-specified specification.

17.4.2 TRANSFORMATION (T)

T is the *transition* function mapping $Q \times P$ to Q . That is, $T(S, p)$ is a process state for each process state S and production rule p . An *execution* is a series of process states $S_0, S_1, S_2, \dots, S_n$, such that $T(S_i, p) = S_{i+1}$. At the beginning of the Gear Box design process, the initial *process state* exists in pure presumed specifications (“design a mass production device, which can be used for lifting light objects or opening a garage door in a family or small warehouse”), and the artifact part and validated specifications have a null or an empty content.

Examples for the *transformation* T are:

- Suppose that the current process state includes the specification “insure a long work life for the rope”, and the production rule p “IF the over-bending of the rope is avoided THEN a long work life for the rope is insured” is included in the designer’s knowledge body. Then, the transformation T alters the current specification to the new requirement “the over-bending of the rope is avoided”.

- Suppose that the current process state includes the specifications (1) $n_0 = 12V / \pi d_{dr}$; (2) $V=30$ feet per hour; and (3) $d_{dr} = 3.75$. Then, the transformation T modifies the artifact part with the new structural attribute $n_0 = 30.57$ rpm.

- Suppose that the current process state includes the structural attributes (1) $P=0.16$; (2) $N_w = 1$; (3) $\lambda = 7^\circ$; (4) $\phi = 20^\circ$, and the specifications (1)

$N_g = N_w * R$; (2) $l = N_w P$; (3) $F_g = 1.125 * \sqrt{(D_{wo} + 2c)^2 - (D_{wo} - 4a)^2}$.

Then the transformation T modifies the artifact part with the new structural

attributes $N_g = 50$ and $F_g = 0.4941$.

- Suppose that the current process state includes the structural attribute “the wire rope diameter = 1/4 inch”; and the specification attribute “the drum diameter should be 15 to 20 times the rope diameter”. Then the transformation T modifies the artifact part with the new structural attribute “drum diameter = 3.75 inches”. Notice that the revised artifact part: <“wire rope diameter=1/4 inch”, “drum diameter = 3.75 inches”> satisfies (logically) the specification part <“the drum diameter should be 15 to 20 times the rope diameter”>.

The design process terminates if the artifact part is fully specified, and the design solution satisfies the specifications. For example, the Gear Box design process is terminated when the transmission parts, casing, shaft set, and accessories are fully specified, and all user-specified requirements (duration, capacity), strength constraints and heat balance are satisfied by the current solution.

17.5 A METHODOLOGY FOR VARIATIONAL DESIGN

In many mechanical systems (such as the Gear Box design), the mathematical model can be characterized by m nonlinear equations in n unknowns. The m equations could be either *equality* constraints (specifying the values of higher-order attributes, such as volume, mass, maximum stress), or *active inequality constraints* (a set of qualitative specifications). In either case, the mathematical model consists of $(n-m)$ degrees of freedom, $(n-m)$ unknowns must be specified and the remaining n variables are computed using the n equations. It is the designer’s job to modify the constraints and specifications however they feel is necessary to find a satisfactory design; it is the purpose of a constraint solver (e.g., Newton-Raphson) to solve simultaneously the entire system of new constraints. An illustration of this methodology is provided in Chapter 14, where the COAST (COnsistency through Analysis of Solution Trajectories) methodology is developed for maintaining design consistency in variational design. COAST is used to maintain a consistent solution to every design modification the designer performs. In this section, we illustrate via the Gear Box example this procedure of revising the entire system of constraints and re-solving them. This methodology can also be considered an instance of the general evolutionary process model presented in Chapter 6.

17.5.1 THE GENERAL METHODOLOGY

In our methodology, a design is represented by a fully-constrained system of equality constraints (denoted by \underline{E}) as defined by the user, and a set of qualitative specifications in the form of inequality constraints (denoted by \underline{Q}). The equality constraints are defined in terms of the dimensions of the part to be created. The

system is solved simultaneously to obtain the desired values of the dimensions. Q are generated by the user to define the successful termination of the process. F defines higher-order attributes (defined in terms of the dimensions) that express higher-level characteristics of the part.

In terms of the general evolutionary process model, the design part includes of all the dimensions of a part. The specifications part consists of any higher-order attributes values, the equality and inequality constraints. At the beginning of the design, the user has a set of qualitative specifications describing the successful termination of the design process (e.g., “produce for less than a given cost” and/or “have a maximum load greater than some value”), and a set of constraints that, when solved, produces a design that may or not satisfy them. If not, the user modifies either the constraints or their values (i.e., a new system of constraints is defined), and/or the dimensions (i.e., solving the entire system of constraints) until the design is satisfactory.

17.5.2 DEMONSTRATION

In order to demonstrate these techniques, a wormgear reducer is designed. The constraints model of the wormgear reducer is given as follows. There are nine dimensions (design parameters) which describe a wormgear assembly:

- N_i = Input Speed (RPM)
- V = Lifting Speed (fpm)
- d = Drum Diameter (in)
- F = Lifting Load (lb.)
- η = Efficiency
- N_w = Number of Threads
- P = Pitch Angle (deg)
- ϕ = Pressure Angle (deg)
- λ = Lead Angle (deg)

From those nine dimensions, twenty higher order attributes (constraints) are defined:

$$\text{Output Speed:} \quad n_o = \frac{12V}{\pi d_{dr}} \quad (46)$$

$$\text{Reduction Ratio:} \quad R = \frac{n_i}{n_o} \quad (47)$$

$$\text{Output Power:} \quad P_o = \frac{FV}{33,000} \quad (48)$$

Input Power: $P_i = \frac{FV}{33,000\eta}$ (49)

Worm Lead: $l = N_w P$ (50)

Worm Pitch Diameter: $D_w = \frac{N_w P \cot(\lambda)}{\pi}$ (51)

Wormgear Teeth Number: $N_g = \frac{\pi d N_i N_w}{12V}$ (52)

Worm Outside Diameter: $D_{w0} = 0.6366P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi}$ (53)

Wormgear Pitch Diameter: $D_g = \frac{d P N_i N_w}{12V}$ (54)

Center Distance: $C = 0.5 \left(\frac{d P N_i N_w}{12V} + \frac{N_w P \cot(\lambda)}{\pi} \right)$ (55)

Wormgear Outside Diameter: $D_{g0} = \frac{0.083333d P N_i N_w}{V} + 0.6366P \cos(\lambda)$ (56)

Length of Threaded Portion:

$$F_w = \sqrt{0.20263P^2 + \frac{0.1061dP^2 N_i N_w \cos(\lambda)}{V} + 0.20263P^2 \cos(2\lambda)}$$
 (57)

Wormgear Face Width:

$$F_g = 1.125 \sqrt{\left(0.6366P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi}\right)^2 - \left(0.764P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi}\right)^2}$$
 (58)

Tangent Force: $F_t = \frac{3.81818\pi FV}{N_i N_w P}$ (59)

Normal Force: $F_n = \frac{3.81818\pi FV \csc(\lambda) \sec(\phi)}{N_i N_w P}$ (60)

Radial Force: $F_r = \frac{3.81818\pi FV \csc(\lambda) \tan(\phi)}{N_i N_w P}$ (61)

Thrust Force: $T_{thrust} = \frac{3.81818\pi FV \cot(\lambda)}{N_i N_w P}$ (62)

Wormgear Pitch Line Velocity: $V_{fg} = \frac{N_i N_w P}{12}$ (63)

Dynamic Load: $F_d = 0.000265152 F\pi + \frac{3.81818\pi FV}{N_i N_w P}$ (64)

Wormgear Tooth Bending Stress:

$$S = \frac{0.88889P \left(0.000265152F\pi V + \frac{3.81818\pi FV}{N_i N_w P} \right)}{Y \sqrt{\left(0.6366P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi} \right)^2 - \left(0.764P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi} \right)^2}} \tag{65}$$

17.5.3 DESIGN EXECUTION

The design process begins with a set of qualitative design specifications as shown in Table 17.7. A five-step scenario is constructed as follows:

Step 1. The design process begins by entering an initial, reasonable value for each dimension: $N_i=1500$, $V=30$, $d=3.75$, $F=150$, $\eta=0.7$, $N_w=1$, $P=0.16$, $\phi=20$, $\lambda=7$.

Step 2. From these dimensions, the higher-order attributes are calculated (from Formulas 46-65) for the designer to examine. The output power P_o is found to be too small. The designer sets $P_o=0.225$, and unconstrains the efficiency, η .

Step 3. Equations (46)-(65) are solved, and the efficiency is found to be $\eta=0.81$. The closest available motor is $\eta=0.8$. Therefore, the designer sets $\eta=0.8$, and unconstrains the output power P_o .

Step 4. Now then, with a higher efficiency, the wormgear pitch diameter D_g is too high. The designer decreases the wormgear pitch diameter, increases the required thrust force, and unconstrains the lead angle and pitch angle.

Step 5. Solving Constraints (46)-(65), the designer finds that the set of qualitative design specifications are satisfied. Therefore, the design execution is terminated with a successful design solution (set of dimensions).

Table 17.7 The Qualitative Design Specifications

Q	≥	q
- η		-0.81
P_o		0.225 (hp)
D_g		-2.6
-S		-12000 (psi)
F_{thrust}		1700 (lb)
-V		-30

REFERENCES

1. Shigley, J. E., *Mechanical Engineering Design*, McGraw-Hill Book Company (5th Edition), 1989.
2. Braha, D., Maimon, O. and Ben-Gal, I., "Case-Studies in General Design Theory: Measurement Tool and Wormgear Reducer," *Technical Report 8-95*, ADMS Lab, Boston University, 1995.

CHAPTER 18

ADAPTIVE LEARNING FOR SUCCESSFUL FLEXIBLE MANUFACTURING CELL DESIGN: A CASE STUDY

In Chapter 13 we presented a method (the P-learning algorithm) for adaptive learning of successful designs that is based on the use of statistical experimental design and stochastic search algorithm. This chapter involves a real industrial problem of designing a flexible manufacturing system that was solved based on the P-learning algorithm.

18.1 INTRODUCTION

Perhaps the greatest overall benefit of using simulation to design and optimize manufacturing environment is that it allows an engineer to obtain a *system-wide view* of the effect of “local” changes to the manufacturing system. If a change is made at a particular work station, its impact on the performance of the overall system may not be predictable by simple formal analyses, therefore resorting to a computer simulation model.

In Chapter 8 we provided a rational means for quantifying how well a proposed design satisfies the governing requirements in terms of its overall *probability* of successfully achieving the functional requirements. In Chapter 13 we presented a method for adaptive learning of successful designs that is based on the use of statistical experimental design and optimization techniques. In this chapter a detailed case study involving the design of a flexible manufacturing cell is presented in order to augment the illustrative example presented in Chapter 13. The system to be designed is an automated design and manufacturing system (ADMS), which has been replacing the traditional job shop-flow shop manufacturing systems. The advantage of ADMS over the job-flow shop manufacturing system is the flexibility of the system and the integration of the diverse functions of manufacturing such as production control and quality control. The system runs automatically and human interference is required only in case of machine failures and initial setups.

The chapter is organized as follows. The physical configuration of the ADMS laboratory is provided in Section 18.2, followed in Section 18.3 by a description of the functional requirements and parameters included in the study. A detailed solution

of the ADMS design problem using the P-learning algorithm introduced in Chapter 13 is given in Section 18.4. Section 18.5 summarizes the chapter.

18.2 PHYSICAL CONFIGURATION

In this section we describe an overview of the Automated Design & Manufacturing Laboratory (ADMS) at Boston University. The ADMS laboratory cell is comprised of a set of *stations* located around a *conveyor* as shown in Figure 18.1. For a detailed schematic description see Figure 18.2. The description of each component is described as follows:

- Design Stations - the design stations contain a PC loaded with several software tools: design packages, production tools' and production system's simulation and control.
- Control Station - the control station contains the central computer and controller, the PLC (Programmable Logic Controller) and its terminal. It does not contain a robot or any production devices.
- CNC Station - the CNC station contains the CNC computer and controller, the CNC machines (one milling machine and one lathe), an ERV7 Robot, sensors, buffers, and a small linear conveyor.
- Quality Control Station - the QC station contains the QC computer and controller, one milling CNC machine, an ERV9 Robot, a vision system, sensors, and buffers. The devices are used for inspection of part quality based primarily on machine vision.

The following elements are common to both the CNC and the QC stations:

- an ACL controller which coordinates the operation of the station and controls the robot and any peripheral devices.
- a PC for user interface to the station controller and device controllers or other software.
- devices that perform an operation in the production cycle, such as CNC machines.
- a SCORBOT-ER7/9 robot which performs material handling tasks such as inserting a part into a CNC machine, loading it onto the pallets, etc.
- a peripheral device which aids the robot in material handling tasks, such as the linear slide-base that supports the robot in the CNC station.

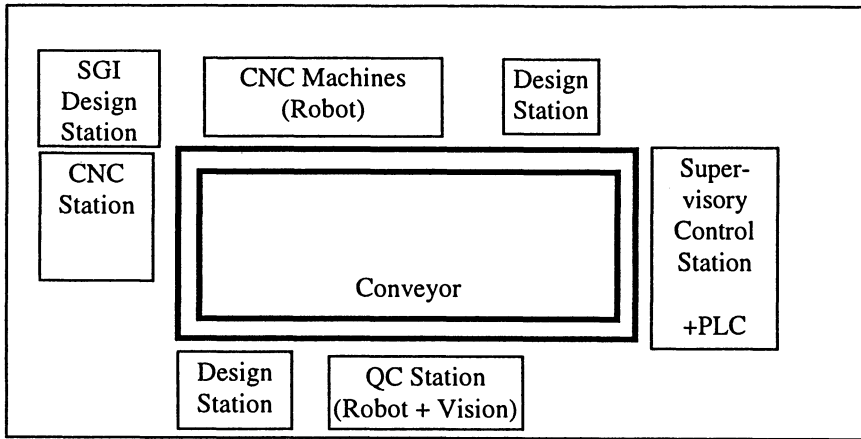


Figure 18.1 ADMS Schematic Configuration

The ADMS combines processes to produce two part types. Figure 18.3 shows the process plan of each part type. The two part types are produced by the two machines in the CNC station, sent to the QC station, and reworked in case of defective units. After the final inspection, parts are placed in the finished parts cart and are assembled by ERV9 Robot to form the final product.

The following information and data as summarized in Tables 18.1, 18.2 have been collected on the system of interest and used to specify operating procedures and probability distributions for the random variables used in the simulation model of the ADMS.

Table 18.1 Probability Distributions of Processing Times (in minutes)

	Part-type 1		Part-type2	
	Distribution	Parameters	Distribution	Parameters
Interarrival Time	Normal	N(8,1)	Normal	N(8,1-2)
Load Time at Robot1	Exponential	Exp(1.2)	Exponential	Exp(1.5)
Unload Time at Robot1	Exponential	Exp(2.0)	Exponential	Exp(1.2)
Processing Time at TMC1	Exponential	Exp(3.5)	Exponential	Exp(4.0)
Load Time at Robot2	Exponential	Exp(0.7)	Exponential	Exp(1.0)
Unload Time at Robot2	Exponential	Exp(2.5)	Exponential	Exp(0.7)
Conveyor speed	Deterministic	25	Deterministic	25
Inspection Time	Exponential	Exp(2.0)	Exponential	Exp(3.0)
Rework Time at TMC2	Exponential	Exp(6.0)	Exponential	Exp(6.0)
Assembling time at Robot2	Exponential	Exp(4.5)	Exponential	Exp(4.5)

Table 18.2 Rework Probabilities

	Part-type 1	Part-type 2
Probability of passing an inspection	0.7	0.6
Probability of failure	0.3	0.4

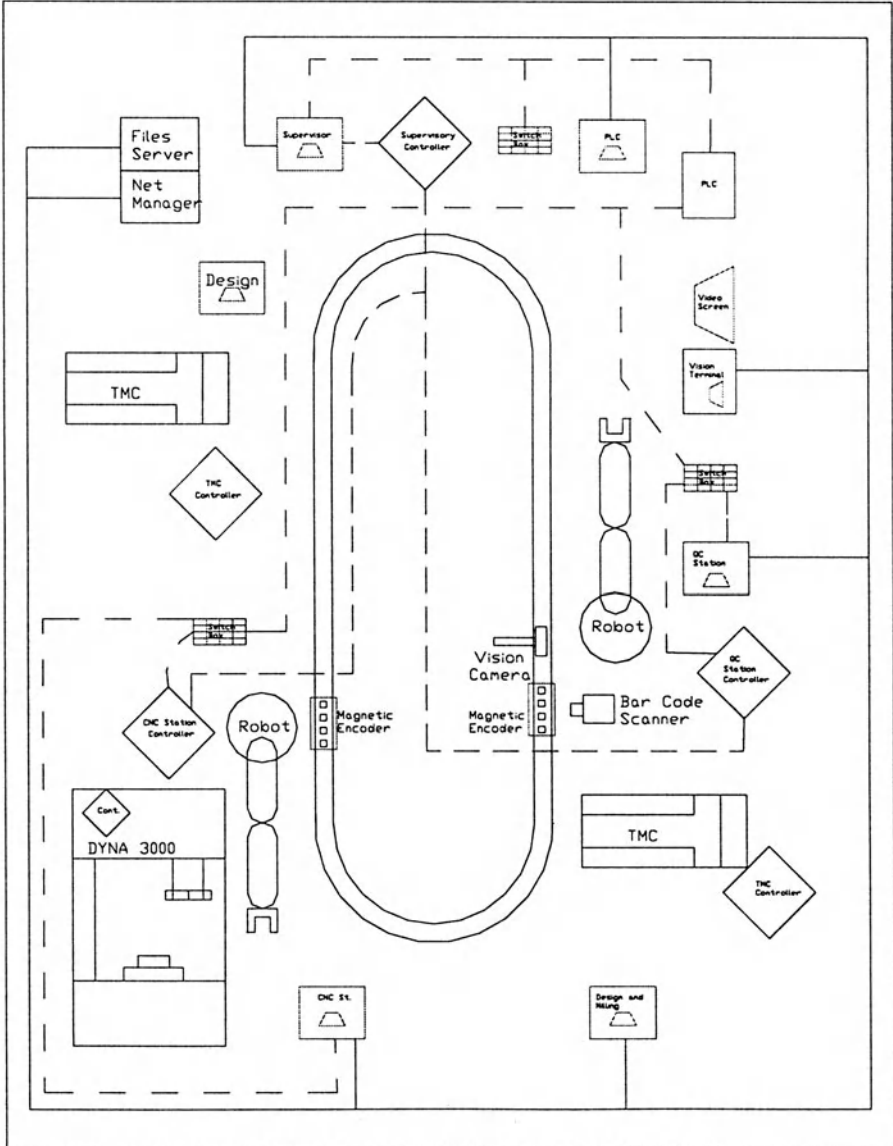


Figure 18.2 ADMS Laboratory: Detailed Schematic Configuration

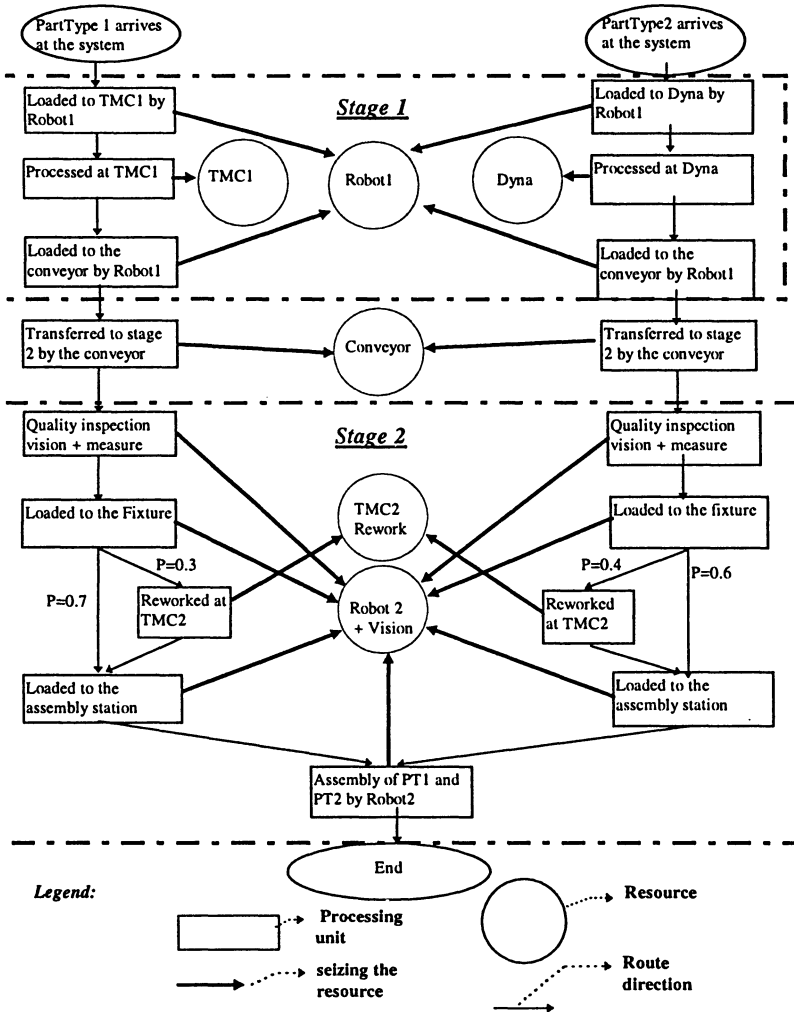


Figure 18.3 Process Plan Logic

18.3 PARAMETERS AND PERFORMANCE MEASURES

18.3.1 PERFORMANCE MEASURES

The decision as to which parameters and structural assumptions compose the ADMS design depends on the goals (performance measures) of the study. In our case, we consider two responses or performance measures of interest.

- *Work in process (WIP)* - the amount of material found in the manufacturing system - in machines, in storage areas, in the transportation subsystem, in inspection stations - at any time. The designer wants it to be as small as possible for a variety of reasons: (1) inventory costs money to create but generates no revenue, (2) the more the inventory the longer it takes to produce the product, and therefore the longer the factory makes the customer wait, (3) the more time items spend in the factory, the more they are vulnerable to damage, (4) many faulty parts are made before a problem is detected, and (5) the space, needed for inventory and material handling system costs money. Since there are 45 free fixtures in the cell, 2 robots, and 3 machines, the designer decides to state his/her first functional requirement as follows: "achieve a work in process level of not more than 51 units."
- *Flow Time (R_{sys})* - the amount of time that a final product (composed of the two part types) spends in the manufacturing system. It is also called throuput time or lead time. The designer wants it to be as small as possible. Since both part types are assembled to form the final product, R_{sys} measures how well the production rates of both part types are balanced. According to experience and design knowledge, the designer decides to state her/his second functional requirement as follows: "achieve a flow time of not more than 250 minutes."

18.3.2 PARAMETERS AND STRUCTURAL ASSUMPTIONS

Parameters can be either *quantitative* or *qualitative*. Quantitative parameters naturally assume numerical values, while qualitative parameters typically represent structural assumptions that are not naturally quantified. We can also classify parameters as being *controllable* or *uncontrollable*. To find out which of possibly many parameters and structural assumptions have the greatest effect on the performance measures presented above (R_{sys} and WIP), the designer collected the following information:

- Since the machines' processing times are already set at their best levels, and further improvement is achievable only by replacing the machines, the machines' processing times are considered fixed aspect of the system and cannot be reduced. However, the rework processing times (countersinking and tapping) can be reduced by using different types of raw material (aluminum versus steel). If the designer decides to use aluminum, then the rework processing time will be random variable from exponential distribution with mean 5.5. If the designer decides to use steel, then the rework processing time will be random variable from exponential distribution with mean 6.5.
- Two types of maintenance plans are considered for the robots. Both plans differ in price and performance. The first maintenance plan assures a better reliability but requires more maintenance time. Reliability is measured in terms of mean time between failures (MTBF) and mean time between repairs (MTTR). The reliability of each CNC machine is considered fixed aspect of the system.

- The conveyor transports the parts from station to station. The conveyor’s speed can be controlled easily. The designer suspects that the conveyor’s speed may affect the performance measures.
- Both robots can perform material handling tasks such as placing a part on the conveyor, removing a part from a feeder, or inserting a part in a CNC machine. These material handling tasks can be performed according to several scheduling procedures. Three policies are considered to be of interest: (1) first come first serve (FCFS), (2) high value first (HVF), which means that the second part type 2 has priority over part type 1, and (3) low value first (LVF), which means that part type 1 has priority over part type 2.
- The interarrival times of raw materials (of both part types) are random variables from normal distributions with mean 8. The average interarrival times of raw materials (of both part types) are considered fixed aspect of the system, while the variance of the interarrival times is considered as uncontrollable parameter. However, the designer considers this uncontrollable factor to be of interest, since he/she wants to assess how a reduction of the variance would increase the performance measures by being able to deal with unforeseen events.
- The Dyna lathe machine can operate in two possible modes: (1) “large stack” mode, which requires longer setup time and longer time between consecutive setups (150min/20min), and (2) “little stack” mode, with requires shorter setup and shorter time between consecutive setups (75 min/10 min).
- The designer would also like to determine whether the parameters interact with each other, i.e., whether the effect of one parameter depends on the levels of the others.

Based on the above information the designer decides to choose *eight* parameters for further consideration, each of which has just *two* levels, as summarized in Table 18.3. The *design problem* to be solved is to determine the optimal combinations of controllable parameters levels that can satisfy the functional requirements (in terms of probability).

Table 18.3 Parameters and their Levels

Parameters	Level 1	Level 2
Raw Material Type (RM)	Steel	Aluminum
Reliability of Robot 1 (R1)	75/4	125/3
Reliability of Robot 2 (R2)	75/4	125/3
Material Handling Policy of Robot 1 (PR1)	FCFS	LVF
Material Handling Policy of Robot 2 (PR2)	FCFS	HVF
Conveyor Speed (CS)	20	30
Dyna lathe machine Operation Mode (DY)	150/20	75/10
Variance of interarrival times of raw materials (ARR)	1	4

As mentioned before, the last parameter (ARR) is uncontrollable (a “noise” parameter). The main reason to include it in further considerations is to try to identify interactions that might exist between the ARR and other controllable parameters. If such interactions exist, the robustness of the system can be increased by controlling the interacting controllable parameters.

18.3.3 EVALUATION OF THE RESPONSES THROUGH SIMULATION

The performance measures were investigated through simulations. The logic for the simulation model is given in Figure 18.3. The system runs automatically and human interference is required only in case of failures and initial setups. By failures we mean any machine failures, identification ambiguities, and faulty operations, all of which are accounted for by the simulation model. For example, failure occur when parts are not properly positioned on the machines, the vision system fails to identify which object is on table, the network ports are detached, and a machine tool is broken. In case of machine (or robot) failure, we assume that the processing of the part is resumed as soon as the machine (or robot) is repaired. Queuing capacity limitations are also accounted for by the simulation model. The simulation model was written using the SIMAN V simulation language [1]. The model consists of more than 100 blocks, more than 2000 entities, and is executed for a horizon of 2000 minutes. In order to lower the possibility of correlated simulation runs, different seeds and streams were used for different executions.

18.4 SOLVING THE DESIGN PROBLEM USING THE P-LEARNING ALGORITHM

The ADMS design problem is described as follows. The parameter index set is $I = \{1, 2, 3, 4, 5, 6, 7, 8\}$, the level index set is $J = \{1, 2\}$, and the required functional requirements (tolerances) are given by $T = \{t_1, t_2 / t_1 \leq 250, t_2 \leq 51\}$. t_1 denotes the flow time, and t_2 denotes the work in process level. The design space includes $2^8 = 256$ feasible design solutions. The sample reduction ratios are set to $\delta_1 = 1/16$, $\delta_2 = 3/8$, and the termination criterion is set to $C = 22$ simulation runs.

- Initialization: In addition to the main parameters in I , the designer decides to determine whether the parameters interact with each other. According to experience and design knowledge, the designer considers the following seven interactions to be of interest: R2/PR2, R2/RM, ARR/PR1, ARR/R1, ARR/CS, ARR/DY, ARR/RM. Based on the reduction ratio δ_1 , the initial representative sample Ω^1 is set to be the L_{16} orthogonal array (see Table 18.4). Since the number of rows (16) equals the required degrees of freedom (the sum of degrees of freedom of main parameters and interactions), the selected orthogonal array is adequate. The L_{16} orthogonal array

guarantees a resolution level of 2, which means that all main parameters are unconfounded with two-parameter interactions, but some groups of two-parameter interactions are confounded (mixed) with each other. There are several ways to assign parameters to the columns of the L_{16} orthogonal array. The assignment process, which is knowledge driven, is performed by a family of linear graphs [1]. The design points in Ω^1 were simulated, each of which for five replications (i.e. $\omega=5$). For each parameter $i \in I$ and each level $j \in J$, the preference probabilities $P(\lambda_i^j \in d)$ are set to 0.5.

Table 18.4 The Initial L_{16} Orthogonal Array (OA)

	ARR	R2	ARR /R2	RM	PR2	R2/ RM	R2/ PR2	CS	ARR /CS	R1	ARR /R1	DY	ARR /DY	PR1	ARR / PR1
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	1	1	1	2	2	2	2	1	1	1	1	2	2	2	2
4	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
5	1	2	2	1	1	2	2	1	1	2	2	1	1	2	2
6	1	2	2	1	1	2	2	2	2	1	1	2	2	1	1
7	1	2	2	2	2	1	1	1	1	2	2	2	2	1	1
8	1	2	2	2	2	1	1	2	2	1	1	1	1	2	2
9	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
10	2	1	2	1	2	1	2	2	1	2	1	2	1	2	1
11	2	1	2	2	1	2	1	1	2	1	2	2	1	2	1
12	2	1	2	2	1	2	1	2	1	2	1	1	2	1	2
13	2	2	1	1	2	2	1	1	2	2	1	1	2	2	1
14	2	2	1	1	2	2	1	2	1	1	2	2	1	1	2
15	2	2	1	2	1	1	2	1	2	2	1	2	1	1	2
16	2	2	1	2	1	1	2	2	1	1	2	1	2	2	1

- The parameters (including two-parameter interactions) that have the greatest effect on the performance measures are identified by applying analysis of variance and pooling up techniques. For both performance measures (flow time and work in process), we conclude with four significant parameters, R1, R2, ARR, DY as shown in Table 18.5 and Table 18.6, respectively. The results in Tables 18.5, 18.6 are based on the assumption that the uncontrollable parameter ARR is assigned to its levels in equal probabilities.

Table 18.5 Final ANOVA Analysis for Flow Time

*** ANALYSIS OF VARIANCE ***					
Flow Time (Rsys)					
by R1, R2, ARR & Interaction DY_ARR					
Source of Variation	Sum of Squares	DF	Mean Square	F	Sig. of F
Main Effects	2447.5	4	611.87	16.092	.000
R1	441.0	1	441.0	11.59	0.06
R2	1444.0	1	1444.0	37.977	.000
ARR	380.25	1	380.25	10.001	.009
DY	182.25	1	182.25	4.793	.051
Explained	2447.5	4	611.87	16.092	.000
Residual	418.25	11	38.02		
Total	2865.750	15	191.05		

Table 18.6 Final ANOVA for Work in Process

*** ANALYSIS OF VARIANCE ***					
Work in Process (WIP)					
by R1, R2, ARR & Interaction DY_ARR					
Source of Variation	Sum of Squares	DF	Mean Square	F	Sig. of F
Main Effects	104.45	4	26.113	10.795	.001
R1	14.822	1	14.822	6.128	.031
R2	37.823	1	37.823	15.63	.002
ARR	35.403	1	35.403	14.63	.003
DY	16.403	1	16.403	6.781	.025
Explained	104.45	4	26.113	10.795	.001
Residual	26.607	11	2.419		
Total	131.057	15	8.737		

- For each significant and *controllable* parameter $i \in I$ and each level $j \in J$, the estimated mean response $\hat{\mu}_{\lambda_i^j}$ and the estimated variance $\hat{\sigma}_{\lambda_i^j}^2$ are computed as summarized in Tables 18.7, 18.8. The estimated mean and variance which are associated with the parameter DY are computed by assuming that the uncontrollable parameter ARR is assigned to its first and second values with equal probabilities.

Table 18.7 Mean and Variance Analysis for Flow Time (R_{sys})

Summaries of Rsys by levels of R1				
Variable	Value		Mean	Std Dev
R1	1		265.87	13.303
R1	2		255.37	13.015
Summaries of Rsys by levels of R2				
Variable	Value		Mean	Std Dev
R2	1		270.125	9.68
R2	2		251.125	10.45
Summaries of Rsys by levels of interaction DY				
Variable	Value		Mean	Std Dev
DY	1		262.25	13.57
DY	2		254.02	10.67

Table 18.8 Mean and Variance Analysis for Work in Process (WIP)

Summaries of WIP by levels of R1				
Variable	Value	Label	Mean	Std Dev
R1	1		53.92	2.66
R1	2		52.0	3.08
Summaries of WIP by levels of R2				
Variable	Value	Label	Mean	Std Dev
R2	1		54.50	2.22
R2	2		51.42	2.89
Summaries of WIP by levels of interaction DY				
Variable	Value	Label	Mean	Std Dev
DY1	1		53.725	1.52
DY1	2		52.2	2.12

- Following the required functional requirements (tolerances) as given by

$T = \{t_1, t_2 / t_1 \leq 250, t_2 \leq 51\}$, the experimental success probability $P_{\lambda_i^j}$ is computed for each significant parameter $i \in I$ and each level $j \in J$ (see Chapter 13, Section 4) with respect to each functional requirement as given in Tables 18.9, 18.10.

Table 18.9 Experimental Success Probabilities with Respect to Flow Time

Factors	Levels	Experimental Success Probability
R1	1	0.0%
	2	1.6%
R2	1	0.0%
	2	30.8%
Dy	1	0.0%
	2	0.1%

Table 18.10 Experimental Success Probabilities with Respect to Work in Process

Factors	Levels	Experimental Success Probability
R1	1	0.0%
	2	4.8%
R2	1	0.0%
	2	22.8%
Dy	1	0%
	2	1%

- For each significant parameter $i \in I$ and each level $j \in J$, the preference probability is updated by setting $P(\lambda_i^j \in d) = P(\lambda_i^j / r \in t)$ with respect to each functional requirement as given in Tables 18.11, 18.12. For each insignificant parameter (including the uncontrollable parameter ARR) $i \in I$ and each level $j \in J$, the preference probability remains unchanged, i.e. $P(\lambda_i^j \in d)$ is set to 0.5.

Table 18.11 A Posteriori Preference Probabilities with Respect to Flow Time

Significant Parameters	Levels	A posteriori preference probabilities
R1	1	0.1%
	2	99.9%
R2	1	0.0%
	2	100.0%
Dy	1	8.5%
	2	91.5%

Table 18.12 A Posteriori Preference Probabilities with Respect to Work in Process

Significant Parameters	Levels	A posteriori preference probabilities
R1	1	0.26%
	2	99.74%
R2	1	0.01%
	2	99.99%
Dy	1	9.5%
	2	90.5%

- In the next step (i.e. S=2), the designer generate $\delta^2 \cdot |\Omega^1| = 0.375 \cdot 16 = 6$ new designs such that the parameter i is assigned to its j th level with the revised preference probability $P(\lambda_i^j \in d)$. The resulting representative sample Ω^2 is described in Table 18.13. The design points in Ω^2 were simulated, each of which for one replication (i.e. $\omega=1$). For each design point $d_q \in \Omega^2$ its overall success probability P_{d_q} is computed, and the design point d_2 that yields the maximum value is selected as the design solution. The overall probability of successfully achieving both functional requirements (i.e., $T = \{t_1, t_2 / t_1 \leq 250, t_2 \leq 51\}$) is computed to be $94\% \cdot 98\% = 92.12\%$.

Table 18.13 The Design Matrix Ω^2

Design	1	2 (Optimal)	3	4	5	6
R2	2	2	2	2	2	2
R1	2	2	2	2	2	2
ARR	1	2	1	2	1	2
PR1	1	1	1	1	1	1
DY	2	2	2	2	2	2
PR2	1	1	1	1	2	2
CS	2	2	1	1	2	2
RM	1	2	1	2	1	2
R _{sys}	250	235	253	256	250	236
WIP	50.2	45.7	50.8	51.3	48.8	46.7

18.5 CONCLUDING REMARKS

Another specific design methodology that uses a statistical approach was developed by Taguchi [2]. The Taguchi method characterizes the design in terms of four types of parameters (signal, control, scaling, and noise parameters) and a single output response [2]. According to the Taguchi method, orthogonal arrays are used in design of experiments in order to find the average effect of each parameter-level when all other parameters are varied. In the selected design solution, each parameter is set to the level that yields the greatest average effect. To evaluate the adaptive approach demonstrated in this chapter, the Taguchi method has been used to aid in the selection of parameter levels that form a solution for the ADMS design problem. For both performance measures, the preferable solution according to the Taguchi method is d_1 in Table 18.13. However, the probability of success ($\approx 78\%$) with respect to the functional requirement of achieving a flow time of not more than 250 minutes, is less than the overall probability of success for the solution selected by the P-learning algorithm (d_2 in Table 18.13). Moreover, the P-learning algorithm yields another solution (d_3 in Table 18.13) that performs better than the solution recommended by the Taguchi method.

To summarize, the P-algorithm for optimizing design decisions is appealing because (1) the adaptive approach offers a homogeneous and unbiased decision criteria for resolving multi-objective design problems, which is a function of their overall probability of success (as introduced in Chapter 8), and (2) the adaptive approach accounts for coupling effects due to highly complex interactions as opposed to the additive approach held by the Taguchi method.

REFERENCES

1. Pegden, C. D., Shannon, R. E., and Sadowski, R. P., *Introduction to Simulation Using SIMAN*, McGraw-Hill, 1990.
2. Ross, P. J., *Taguchi Techniques for Quality Engineering*, McGraw-Hill, 1988.

CHAPTER 19

MAINTAINING DESIGN CONSISTENCY: EXAMPLES

In order to demonstrate the ability to maintain a consistent design by analyzing solution trajectories (as presented in Chapter 14), detailed results from several design examples are given. Between most steps of the design, there are multiple competing solution trajectories that a typical numerical constraint solver could converge. It can be seen, however, that the desired solution follows a unique solution trajectory that can be followed to maintain the consistency. The algorithm flowchart is repeated here to make this chapter as self-contained as possible.

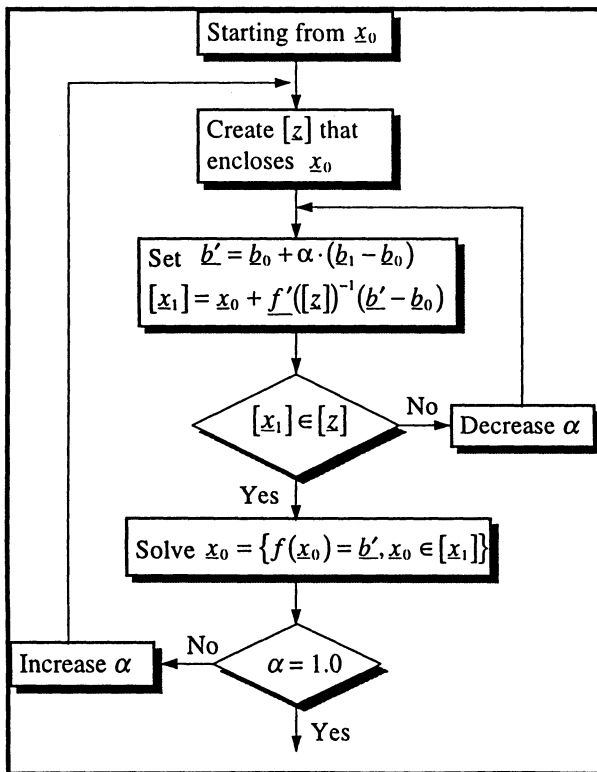


Figure 19.1 COAST Algorithm

19.1 WORMGEAR ASSEMBLY PROBLEM

19.1.1 DIMENSIONS - WORMGEAR ASSEMBLY

The first example is related to the design of a worm and wormgear assembly (see also Chapter 17). There are 9 dimensions which fully describe the wormgear assembly for this problem:

N_i =	Input Speed (RPM)	V =	Lifting Speed (fpm)
d =	Drum Diameter (in)	F =	Lifting Load (lb)
η =	Efficiency	N_w =	Number of threads
P =	Pitch Angle (deg)	ϕ =	Pressure Angle (deg)
λ =	Lead Angle (deg)		

Additionally, there are 19 attributes defined in the constraint model which can also be constrained (see Appendix A).

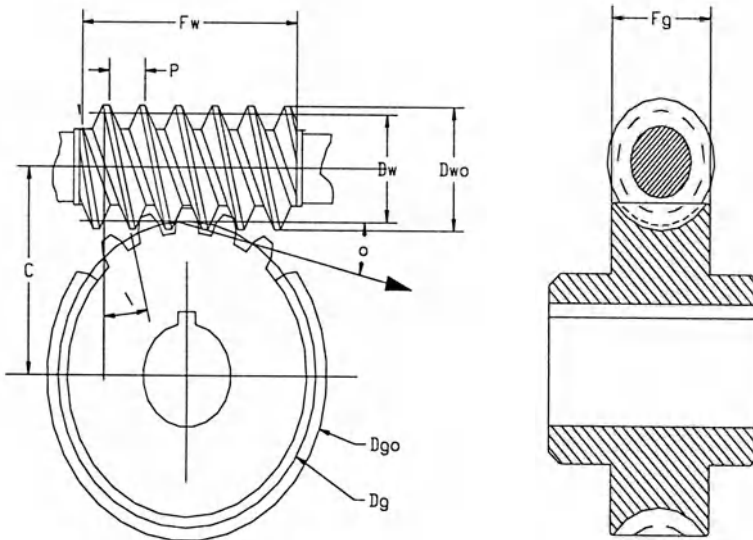


Figure 19.2 Parameters and Attributes of Worm/Wormgear

19.1.2 DESIGN EXECUTION

To test the design consistency methods, a design scenario is constructed and an 8-step solution process is demonstrated. We are designing a worm and worm-gear assembly with the following specifications:

$$\begin{array}{lll}
 F_{thrust}, F_n \leq 2500 & F_t \leq 500 & 1000 \leq N_i \leq 2000 \\
 20 \leq V \leq 50 & F \geq 250 & 0.4 \leq d_{go} \leq 0.5 \\
 1.5 \leq d_{go} + d_{wo} \leq 1.6 & 0.8 \leq f_g \leq 0.9 & \eta = 0.7
 \end{array}$$

Additionally, the worm and worm-gear assemblies are available only in the following dimensions:

λ	4-9 degrees in 0.5 degree increments
ϕ	18-25 degrees in 0.5 degree increments
N_w	1, 2, or 3
P	0.05 - 0.25 in 0.025 inch increments
d	0.05 - 0.25 in 0.05 inch increments

Starting from an initial solution, the constraints are modified at each step until the design is satisfied finally in Step 8. Each step consists of the constraints used to define the design, any solution trajectories from the previous solution to the current solution, the final correct solution, the corresponding attribute values, and comments on the satisfaction of the constraints and what dimensions or attributes to constrain and release.

Step 1:

θ_1 :

$$N_i = 1500, V = 30, d = 0.25, F = 275, \eta = 0.70, N_w = 1, P = 0.1, \phi = 20, \lambda = 7$$

Solution Trajectories:

None.

M_1 :

$$N_i = 1500, V = 30, d = 0.25, F = 275, \eta = 0.70, N_w = 1, P = 0.1, \phi = 20, \lambda = 7$$

Attribute Values:

$$F_n = 5761., F_{thrust} = 5373., F_t = 660, d_{go} = 0.167, d_{wo} = 0.322, f_g = 0.306$$

Comments:

F_{thrust} is too high; F_t is too high; d_{go} is too small; f_g is too small. Constrain F_{thrust} and F_t and release P and λ .

Step 2: $\theta_2 :$

$$N_i = 1500, V = 30, d = 0.25, F = 275, \eta = 0.70, N_w = 1, F_{thrust} = 2000., \phi = 20, F_t = 475$$

Solution Trajectories:

α	P, λ (correct trajectory)	P, λ (wrong trajectory)	P, λ (wrong trajectory)
0	0.1, 0.122173 (rad)	0.1, 3.26377(rad)	0.1, -3.01942(rad)
0.1	0.102881, 0.126659	0.102881, 3.26825	0.102881, -3.01493
0.2	0.105933, 0.131782	0.105933, 3.27338	0.105933, -3.00981
0.3	0.109171, 0.13769	0.109171, 3.27928	0.109171, -3.0039
0.4	0.112613, 0.144576	0.112613, 3.28617	0.112613, -2.99702
0.5	0.11628, 0.152704	0.11628, 3.2943	0.11628, -2.98889
0.6	0.120194, 0.162442	0.120194, 3.30403	0.120194, -2.97915
0.7	0.12438, 0.174319	0.12438, 3.31591	0.12438, -2.96727
0.8	0.128868, 0.189122	0.128868, 3.33071	0.128868, -2.95247
0.9	0.133692, 0.208073	0.133692, 3.34967	0.133692, -2.93352
1.0	0.138891, 0.23318	0.138891, 3.37477	0.138891, -2.90841

 $M_2 :$

$$N_i = 1500, V = 30, d = 0.25, F = 275, \eta = 0.70, N_w = 1, P = 0.139, \phi = 20, \lambda = 13.4$$

Attribute Values:

$$F_n = 2188., F_{thrust} = 2000., F_t = 475, d_{go} = 0.231, d_{wo} = 0.272, f_g = 0.305$$

Comments:

λ is too high; d_{go} is too small; d_{wo} is too small; f_g is too small. Constrain F_{thrust} , d_{go} , and d_{wo} and release V, P, and λ .

Step 3: $\theta_3 :$

$$N_i = 1500, F_{thrust} = 2000, d = 0.25, F = 275, \eta = 0.70, N_w = 1, d_{go} = 0.5, \phi = 20, d_{wo} = 1.0$$

Solution Trajectories:

α	V, P, λ (correct trajectory)	V, P, λ (wrong trajectory)	V, P, λ (wrong trajectory)
0	30.0012, 0.138894, 0.233184	12.4188, 0.122476, 3.25263	12.4188, 0.122476, -3.03056
0.1	34.9059, 0.169918, 0.222152	12.3216, 0.1354, 3.24132	12.3216, 0.1354, -3.04186
0.2	37.9066, 0.196635, 0.208877	12.1023, 0.146066, 3.23245	12.1023, 0.146066, -3.05074
0.3	38.9586, 0.218312, 0.193758	11.825, 0.155099, 3.22523	11.825, 0.155099, -3.05796
0.4	38.3783, 0.23488, 0.177766	11.5219, 0.162896, 3.21921	11.5219, 0.162896, -3.06398
0.5	36.7283, 0.247009, 0.162062	11.2105, 0.169722, 3.21409	11.2105, 0.169722, -3.06909
0.6	34.5593, 0.255736, 0.147511	10.9002, 0.175769, 3.20968	10.9002, 0.175769, -3.07351
0.7	32.2527, 0.262046, 0.134517	10.5963, 0.181175, 3.20581	10.5963, 0.181175, -3.07737
0.8	30.0166, 0.266691, 0.123131	10.3019, 0.186046, 3.2024	10.3019, 0.186046, -3.08078
0.9	27.9431, 0.270192, 0.113229	10.0183, 0.190465, 3.19936	10.0183, 0.190465, -3.08382
1.0	26.0614, 0.272894, 0.104624	9.74625, 0.194498, 3.19664	9.74625, 0.194498, -3.08655

M_3 :

$$N_i = 1500, V = 26.1, d = 0.25, F = 275, \eta = 0.70, N_w = 1, P = 0.273, \phi = 20, \lambda = 6$$

Attribute Values:

$$F_n = 2138., F_{thrust} = 1998., F_t = 210., d_{go} = 0.5, d_{wo} = 0.999, f_g = 0.901$$

Comments:

All attribute values seem to check out fairly well. P is now too big; d_{wo} is a little too small; f_g is a little too big. Constrain F_{thrust} , d_{go} , and d_{wo} , and release N_i , d, and λ .

Step 4:

θ_4 :

$$F_{thrust} = 2000, V = 25, d_{go} = 0.45, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, d_{wo} = 1.1$$

Solution Trajectories:

α	N_i , d, λ (correct trajectory)	N_i , d, λ (wrong trajectory)	N_i , d, λ (wrong trajectory)
0	2191., 0.217413, 0.0834479	2922.02, 0.293384, 3.20423	2922.02, 0.293384, -3.07896
0.1	2216.81, 0.211867, 0.0824806	2947.87, 0.288551, 3.20368	2947.87, 0.288551, -3.07951
0.2	2242.62, 0.206449, 0.0815354	2973.73, 0.283803, 3.20314	2973.73, 0.283803, -3.08004
0.3	2268.44, 0.201154, 0.0806115	2999.58, 0.279136, 3.20261	2999.58, 0.279136, -3.08057
0.4	2294.26, 0.195979, 0.0797082	3025.44, 0.274549, 3.20209	3025.44, 0.274549, -3.08109
0.5	2320.08, 0.190919, 0.0788249	3051.29, 0.270039, 3.20158	3051.29, 0.270039, -3.08161
0.6	2345.9, 0.18597, 0.0779609	3077.15, 0.265606, 3.20108	3077.15, 0.265606, -3.08211
0.7	2371.72, 0.181129, 0.0771156	3103., 0.261246, 3.20058	3103., 0.261246, -3.0826
0.8	2397.54, 0.176393, 0.0762883	3128.86, 0.256958, 3.2001	3128.86, 0.256958, -3.08309

0.9	2423.36, 0.171758, 0.0754785	3154.71, 0.252741, 3.19962	3154.71, 0.252741, -3.08357
1.0	2449.18, 0.16722, 0.0746857	3180.57, 0.248592, 3.19915	3180.57, 0.248592, -3.08404

M_4 :

$$N_i = 2450, V = 25, d = 0.167, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4.3$$

Constrained Attribute Values:

$$F_n = 2123., F_{thrust} = 1990., F_t = 150., d_{go} = 0.45, d_{wo} = 1.095, f_g = 0.877$$

Comments:

N_i is too high; all other dimension values are within range. Change dimension values to match availability.

Step 5:

θ_5 :

$$N_i = 2000, V = 25, d = 0.15, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4$$

Solution Trajectories:

There are no solution trajectories in this step since all the dimensions are defined explicitly.

M_5 :

$$N_i = 2000, V = 25, d = 0.15, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4$$

Attribute Values:

$$F_n = 2796., F_{thrust} = 2621., F_t = 183., d_{go} = 0.368, d_{wo} = 1.17, f_g = 0.909$$

Comments:

F_{thrust} is too high; F_n is too high; d_{go} is too small; f_g is too high. Constrain F_n , f_g , and d_{go} and release d , P , and λ .

Step 6:

θ_6 :

$$N_i = 2000, V = 25, F_n = 2000, F = 275, \eta = 0.70, N_w = 1, f_g = 0.85, \phi = 20, d_{go} = 0.45$$

Solution Trajectories:

α	d, P, λ (correct trajectory)	d, P, λ (wrong trajectory)	d, P, λ (wrong trajectory)
0	0.150002, 0.225007, 0.0698109	0.340516, 0.225007, 3.07178	0.340516, 0.225007, -3.2114
0.1	0.154174, 0.226191, 0.0714829	0.344666, 0.226191, 3.07011	0.344666, 0.226191, -3.21308
0.2	0.158232, 0.227439, 0.0732392	0.3487, 0.227439, 3.06835	0.3487, 0.227439, -3.21483
0.3	0.162167, 0.228757, 0.0750866	0.352609, 0.228757, 3.06651	0.352609, 0.228757, -3.21668
0.4	0.165974, 0.230151, 0.0770328	0.356387, 0.230151, 3.06456	0.356387, 0.230151, -3.21863
0.5	0.169642, 0.231627, 0.0790864	0.360025, 0.231627, 3.06251	0.360025, 0.231627, -3.22068
0.6	0.173163, 0.233191, 0.0812571	0.363513, 0.233191, 3.06034	0.363513, 0.233191, -3.22285
0.7	0.176529, 0.234851, 0.0835559	0.366842, 0.234851, 3.05804	0.366842, 0.234851, -3.22515
0.8	0.179727, 0.236615, 0.085995	0.370001, 0.236615, 3.0556	0.370001, 0.236615, -3.22759
0.9	0.182746, 0.238495, 0.0885887	0.372977, 0.238495, 3.053	0.372977, 0.238495, -3.23018
1.0	0.185574, 0.2405, 0.091353	0.375757, 0.2405, 3.05024	0.375757, 0.2405, -3.23295

M_6 :

$$N_i = 2000, V = 25, d = 0.19, F = 275, \eta = 0.70, N_w = 1, P = 0.24, \phi = 20, \lambda = 5.2$$

Attribute Values:

$$F_n = 2017., F_{thrust} = 1888., F_t = 172., d_{go} = 0.456, d_{wo} = 0.992, f_g = 0.851$$

Comments:

d_{wo} is a little too small; assign available numbers for the dimensions; constrain d_{go} and d_{wo} and release d and λ .

Step 7:

θ_7 :

$$N_i = 2000, V = 25, d_{wo} = 1.1, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, d_{go} = 0.45$$

Solution Trajectories:

α	d, λ (correct trajectory)	d, λ (wrong trajectory)	d, λ (wrong trajectory)
0	0.208848, 0.0841711	0.3993, 3.20464	0.3993, -3.07855
0.1	0.20844, 0.0831159	0.398904, 3.20404	0.398904, -3.07915
0.2	0.208032, 0.0820868	0.398507, 3.20346	0.398507, -3.07973
0.3	0.207624, 0.0810827	0.398111, 3.20288	0.398111, -3.0803
0.4	0.207216, 0.0801028	0.397714, 3.20232	0.397714, -3.08087
0.5	0.206809, 0.0791462	0.397317, 3.20177	0.397317, -3.08142
0.6	0.206402, 0.0782122	0.39692, 3.20122	0.39692, -3.08196
0.7	0.205995, 0.0772998	0.396523, 3.20069	0.396523, -3.08249
0.8	0.205589, 0.0764084	0.396126, 3.20017	0.396126, -3.08302
0.9	0.205182, 0.0755373	0.395729, 3.19965	0.395729, -3.08353
1.0	0.204776, .0746857	0.395332, 3.19915	0.395332, -3.08404

M_7 :

$$N_i = 2000, V = 25, d = 0.2, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4.3$$

Attribute Values:

$$F_n = 2601., F_{thrust} = 2437., F_t = 183., d_{go} = 0.443, d_{wo} = 1.095, f_g = 0.877$$

Comments:

F_n is a little too high; Constrain F_n and release V.

Step 8:

θ_8 :

$$N_i = 2000, F_n = 2450, d = 0.2, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4.5$$

Solution Trajectories:

There is only one solution trajectory for V:

α	V (correct trajectory)
0	26.1604
0.1	26.0085
0.2	25.8566
0.3	25.7048
0.4	25.5529
0.5	25.401
0.6	25.2491
0.7	25.0972
0.8	24.9453
0.9	24.7935
1.0	24.6416

M_8 :

$$N_i = 2000, V = 24.6, d = 0.2, F = 275, \eta = 0.70, N_w = 1, P = 0.225, \phi = 20, \lambda = 4.5$$

Constrained Attribute Values:

$$F_n = 2446., F_{thrust} = 2291., F_t = 180, d_{go} = 0.448, d_{wo} = 1.053, f_g = 0.857$$

Comments:

All dimensions and attributes satisfy the specifications.

19.2 HELICAL COMPRESSION SPRING PROBLEM

Finally, the mechanical design of a helical compression spring is demonstrated. The spring is described using ten variables:

1. σ_{all} = maximum allowable stress
2. G = shear modulus
3. F = safety factor
4. R = coil radius
5. d = wire diameter
6. N = number of coils
7. δ = deflection
8. L_f = free length
9. C = spring index
10. P = load

Additionally, there are four relations given between these variables:

1. $L_f - (\delta + d(N + 2)) = 0$
2. $\delta - \frac{4PC^4N}{GR} = 0$
3. $C - \frac{2R}{d} = 0$
4. $\pi\sigma_{all}d^3 - 16PRF\left(\frac{4C-1}{4C-4} + \frac{0.615}{C}\right) = 0$

θ_0 :

Create a helical compression spring according to the following specifications:

1. $\sigma_{all} = 1.303 \times 10^3$ MPa

2. $G = 7.929 \times 10^4$ MPa
3. $F = 2$
4. $\delta = 12.7$ mm
5. $L_f = 63.5$ mm
6. $P = 444.82$ N

M_0 :

Plugging in the given variable values into the four relations yields a system of four equations and four unknowns (R, d, N, and C). Solving this system of equations shows that there are four mathematically possible solutions:

1. (R, d, N, C) = (6.13, 3.11, 14.33, 3.945)
2. (R, d, N, C) = (-5.36, -3.01, -18.86, 3.5618)
3. (R, d, N, C) = (0.063, 0.204, 246.4, 0.6169)
4. (R, d, N, C) = (-0.0064, -0.206, -248.5, 0.6168)

Of these four solutions, the first one is the desired solution.

θ_1 :

After some time, the specifications are modified. This time, the desired free length is increased to 127.0 mm. Create a helical compression spring according to the updated specifications:

1. $\sigma_{all} = 1.303 \times 10^3$ MPa
2. $G = 7.929 \times 10^4$ MPa
3. $F = 2$
4. $\delta = 12.7$ mm
5. $L_f = 127.0$ mm
6. $P = 444.82$ N

M_1 :

Again, plugging the variables into the relations yields a system of four equations and four unknowns and there are, again, four possible solutions:

1. (R, d, N, C) = (0.092, 0.302, 376.6, 0.6099)
2. (R, d, N, C) = (-3.67, -2.785, -43.05, 2.635)
3. (R, d, N, C) = (3.84, 2.81, 38.7, 2.74)
4. (R, d, N, C) = (-0.0925, -0.3034, -378.7, 0.6097)

The problem is to automatically determine which of the four solutions is the intended solution. It may be tempting to characterize the solutions and determine the correct solution through a set of rules. While this may be possible in isolated circumstances, it is problematic to have to create a rule-base for every different problem. Instead, by using the COAST methodology, design consistency can be achieved in a wide variety

of different problems. First, a homotopy is created between the two systems of equations and the trajectory of the initial solution is followed as it moves to its new value. In this case, similarity between solutions can be expressed as the Euclidean norm of the dimension values. The first iteration of the COAST method is shown in the following two graphs. Each iteration begins from an initial solution, d_0 . An enclosure, $[z]$ is created around the solution and an enclosure is created, $[d_1]$, that is guaranteed to contain the new solution, d_1 , for a given step in the parameter vector. These steps are performed iteratively towards the final solution: $(R,d,N,C) = (3.84, 2.81, 38.7, 2.74)$. More charts of the convergence process for this example are given in Section 19.6.

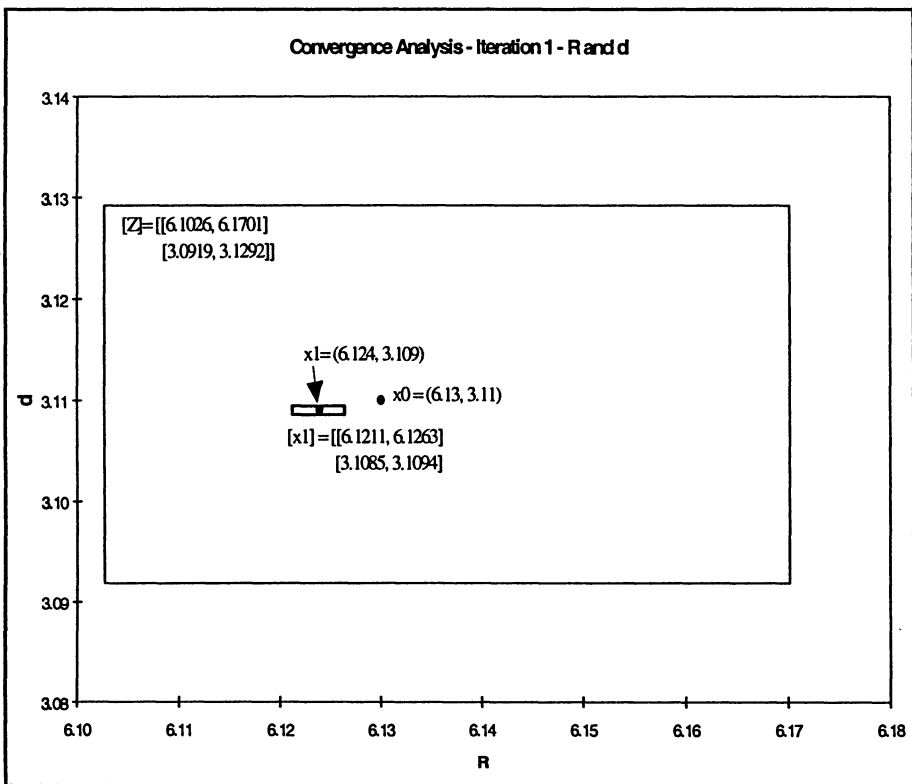


Figure 19.3 Helical Spring Convergence

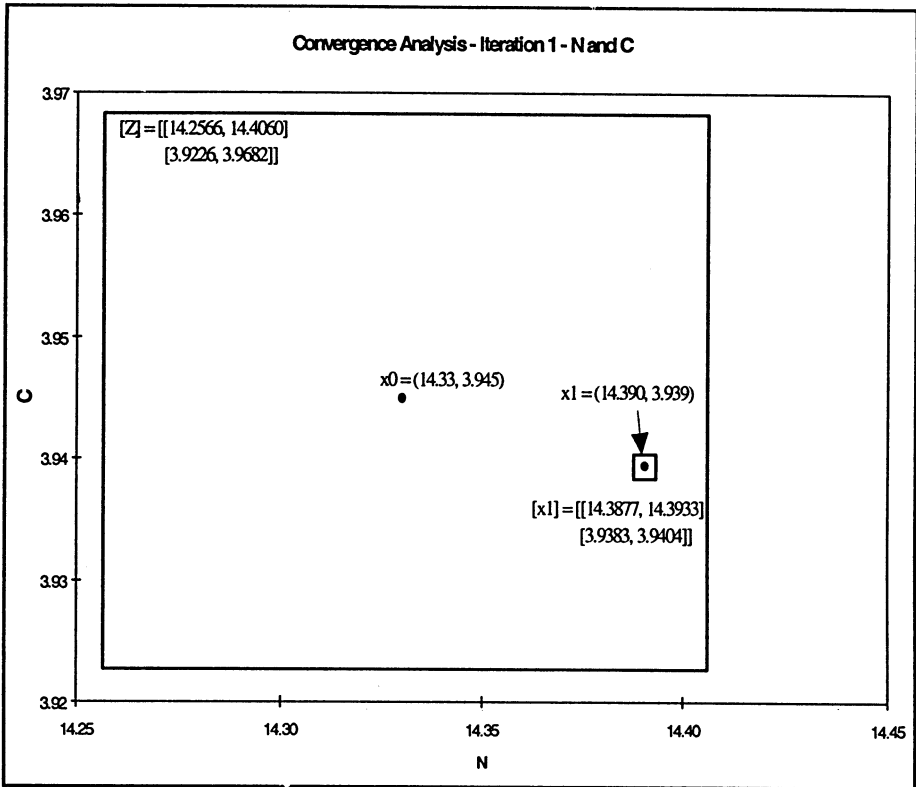


Figure 19.4 Helical Spring Convergence

19.3 OTHER DESIGN AREAS

The importance of design consistency can be seen in other design areas as well. Typically, the designer is presented with a set of requirements that do not fully constrain the design. Hence, the designer uses their expertise to select a “good” solution. Their expertise may include anything from knowledge of their specific design domain to intricacies of their specific situation to such nuances as “office politics”. When the design requirements are eventually modified, the design system should honor the choices the designer made and select the design configuration that is most similar to the original design, while still meeting the new requirements. In order to do this without requiring the designer to explicitly define every constraint and assumption they made, the similarity must be defined between subsequent design iterations.

Consider the field of electrical engineering where the design requirements may be:

Create an active low-pass filter with a pass-band from 0 to 3.47 kHz

(maximum ripple of 0.07 dB) and stop-band starting at 3.8 kHz (minimum attenuation of 50 dB). Assume the OPAMP's are non-ideal.

Given this set of requirements, the designer may design a ninth-order Cauer-parameter active filter (Figure 19.5). Table 19.1 gives the value of each element (resistor values in kΩ, capacitor values in F). When the requirements are modified, clearly any electronic design system should attempt to first find a different ninth-order Cauer-parameter active filter before searching for other types of circuits to satisfy the requirements. And if another type of circuit is needed, it should be chosen to be most "similar" to the original design.

Table 19.1 Element Values for Active Filter

R ₁ : 5.4779	R ₂ : 2.0076	R ₃ : 3.300	R ₄ : 3.300	R ₅ : 4.5898	R ₆ : 4.44
R ₇ : 6.00	R ₈ : 3.300	R ₉ : 3.300	R ₁₀ : 4.2572	R ₁₁ : 3.2201	R ₁₂ : 5.8833
R ₁₃ : 3.300	R ₁₄ : 3.300	R ₁₅ : 5.6260	R ₁₆ : 3.6368	R ₁₇ : 1.0301	R ₁₈ : 3.300
R ₁₉ : 3.300	R ₂₀ : 5.8085	R ₂₁ : 1.2201	C ₁ : 12	C ₂ : 10	C ₃ : 6.8
C ₄ : 10	C ₅ : 4.7	C ₆ : 10	C ₇ : 6.8	C ₈ : 10	C ₉ : 10

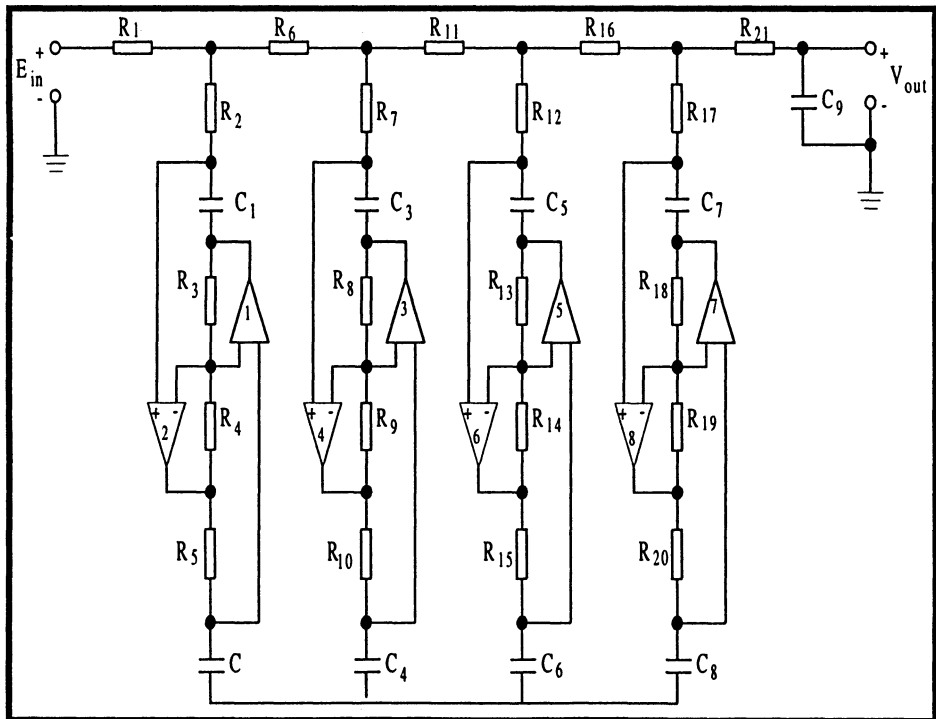


Figure 19.5 Ninth-Order Cauer-Parameter Active Filter

Or consider the area of production planning where a designer may need to position a robotic manipulator to perform as follows:

The PUMA 560 is a 6 DOF robotic manipulator with parameters as shown in Table 19.2. For a given assembly, it is determined from the product design that at a certain point in the assembly process, the PUMA 560 needs to be positioned at the end point $(-0.244, 0.423, -0.555)$ with end effector orientation: $\vec{e}_1 = (-0.94, 0.33, 0)$, $\vec{e}_2 = (-0.03, -0.09, 0.80)$, and $\vec{e}_3 = (-0.33, -0.94, 0.09)$. Calculate the values of the remaining unknown joint parameters.

Table 19.2 Joint Parameters for a PUMA 560

Joint	a_i	d_i
θ_1	0	0.671
θ_2	0.432	-0.15
θ_3	-0.02	0
θ_4	0	0.433
θ_5	0	0
θ_6	0	0

The designer selects the following solution: $\underline{q}_0 = (120 \ 80 \ 30 \ 80 \ 130 \ 158.4)$. Again, when the requirements are modified, the design system should find the manipulator configuration that is the most "similar" to the first configuration.

Finally, consider object-oriented software engineering where a designer may be given the following requirements:

Create an object model of a warehouse system. Parts come into the receiving dock. There, they are placed in pallets and walked around the warehouse, depositing the parts in appropriate bins (located along the aisles in the warehouse). The system should be able to keep track of inventory, be able to give a part's location in the warehouse, and perform the following interactions:

1. Identify new shipments that come into the loading dock, locate their purchase order and the location of the items in the warehouse, and adjust the inventory accordingly;
2. Input new orders and give their location in the warehouse and adjust the inventory accordingly.

To solve this problem, the designer creates the object model shown in Figure

19.6. There are typically many ways to break down a software problem into an object model. In order to make this object model, the designer had to make numerous choices which should be honored by a software design system when the requirements are modified.

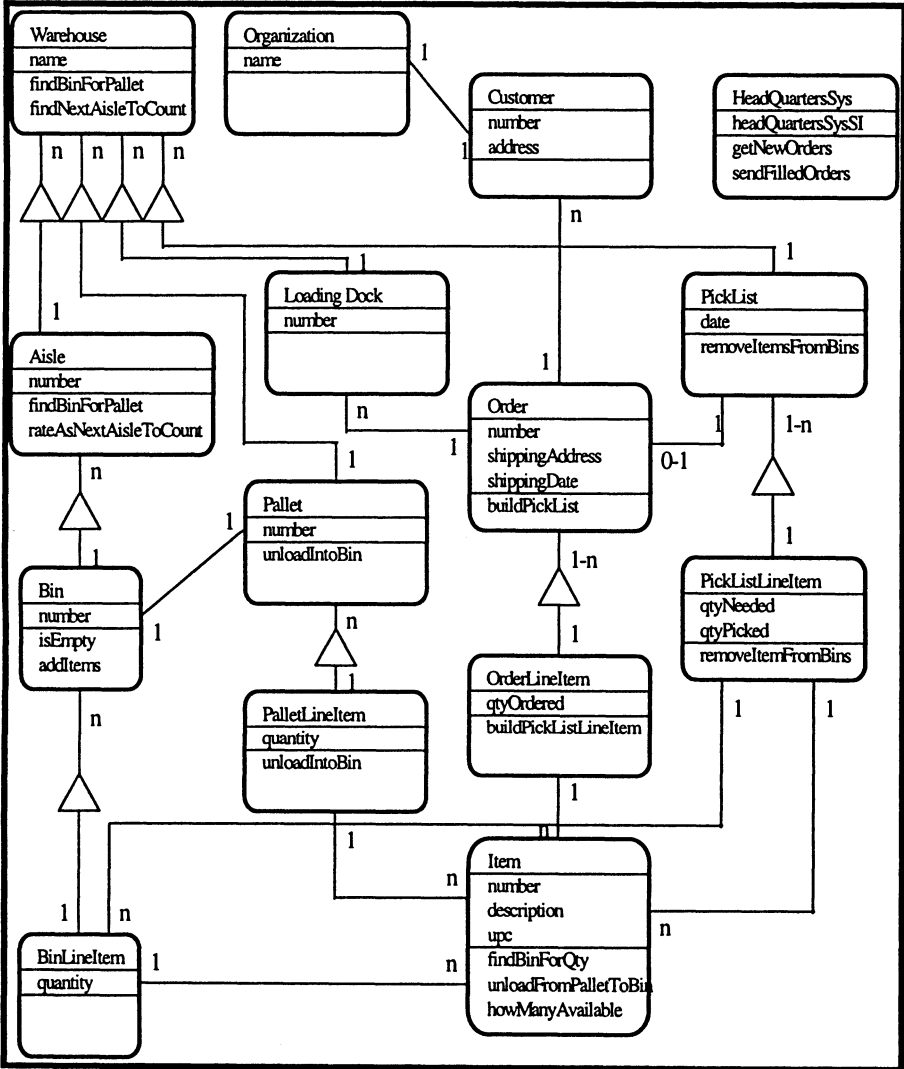


Figure 19.6 Warehouse System Object Model

19.4 POINT AT A DISTANCE FROM TWO POINTS

We demonstrate the situation of creating a point at a certain distance from two other points. The unknown point has two dimensions: the x and y coordinates. Additionally, let there be two attributes: distance from the point at $(0,0)$ and distance from the point at $(3,4)$. The two attributes are each equations in terms of the dimensions: $\sqrt{x^2 + y^2}$ and $\sqrt{(x-3)^2 + (y-4)^2}$, respectively. The user begins by specifying an initial guess of the desired point:

$$\theta_0^C: \quad M_0: \\ \begin{cases} x = 1 \\ y = 4 \end{cases} \rightarrow \begin{cases} x = 1 \\ y = 4 \end{cases}$$

The values of the attributes are calculated and the point is found to be distances 4.12 and 2 from $(0,0)$ and $(3,4)$, respectively. The user then decides that the point should be distances 3 and 6 from $(0,0)$ and $(3,4)$. The two attributes are solved simultaneously yielding the solution, $(-2.51, 1.64)$:

$$\theta_1^C: \quad M_1: \\ \begin{cases} \text{distfrom}(0,0) = 3 \\ \text{distfrom}(3,4) = 6 \end{cases} \rightarrow \begin{cases} x = -2.51 \\ y = 1.64 \end{cases}$$

The two distance attributes at the beginning of the problem can be interpreted as in Figure 19.7-A. When the constraints are modified, however, there are two possible solution (see Figure 19.7-B).

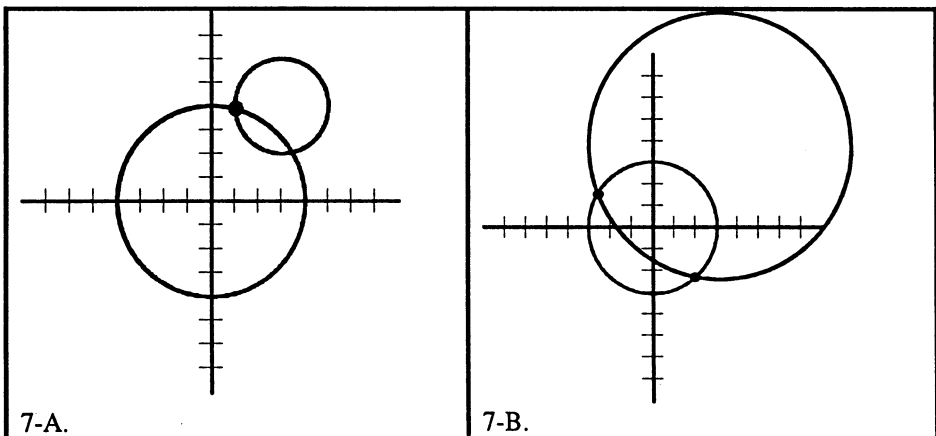


Figure 19.7 Simple Variational Design Example

As the distance radii of the circles are changed, the desired point creates a trajectory from its original position to its new position (as shown in Figure 19.8).

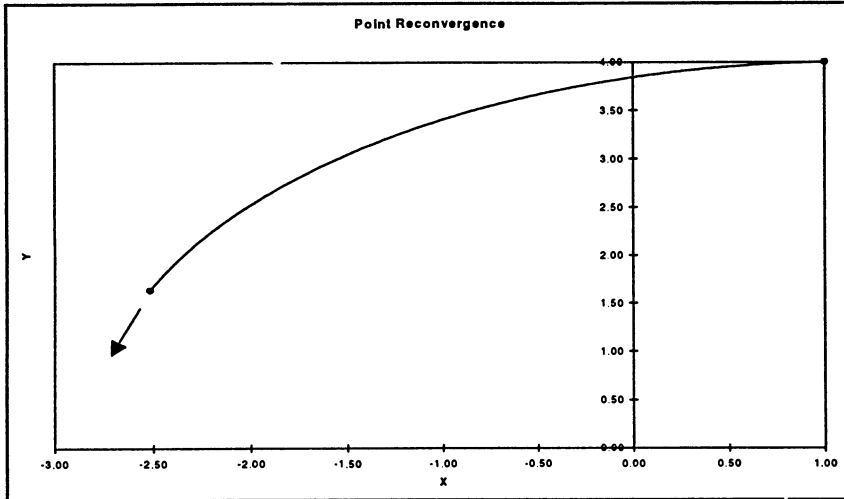
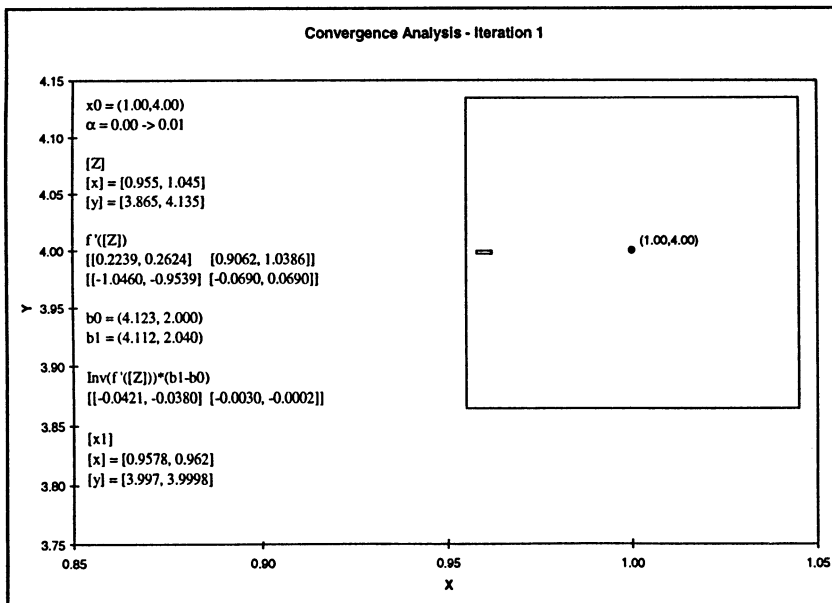
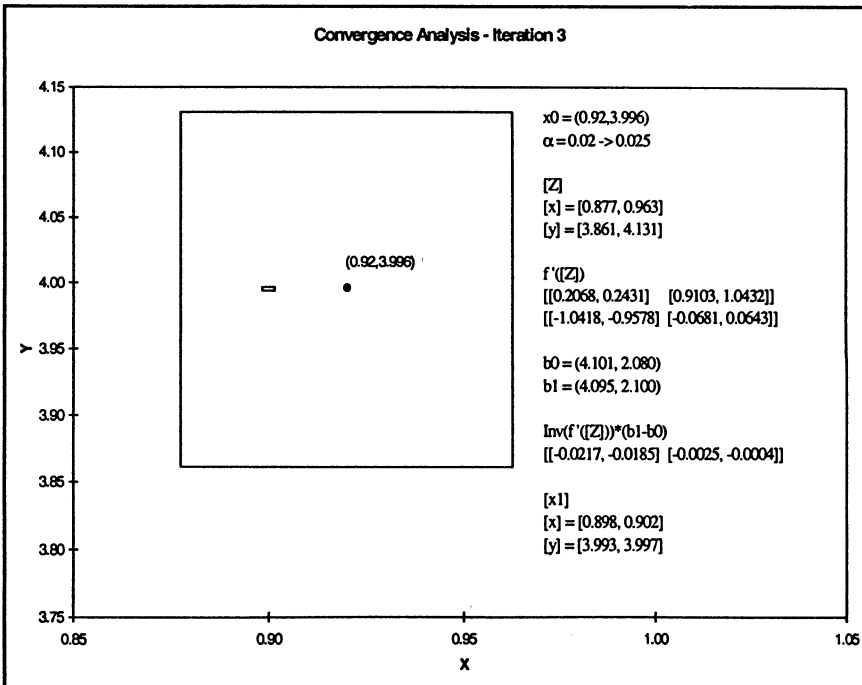
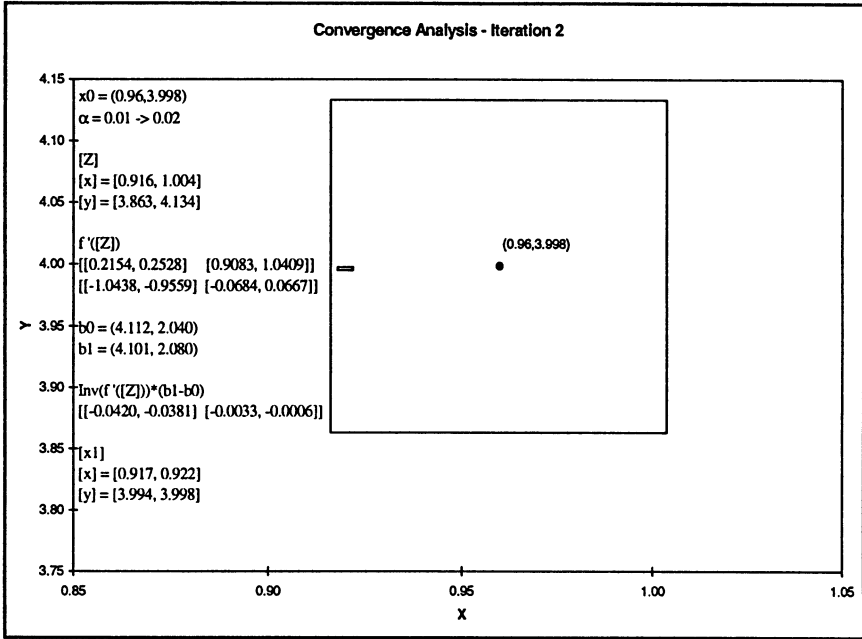
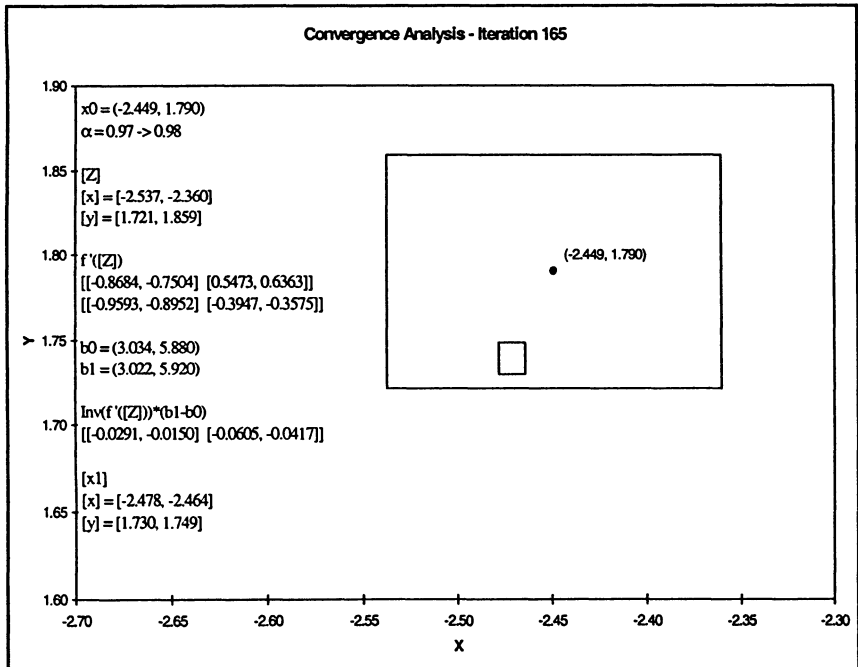
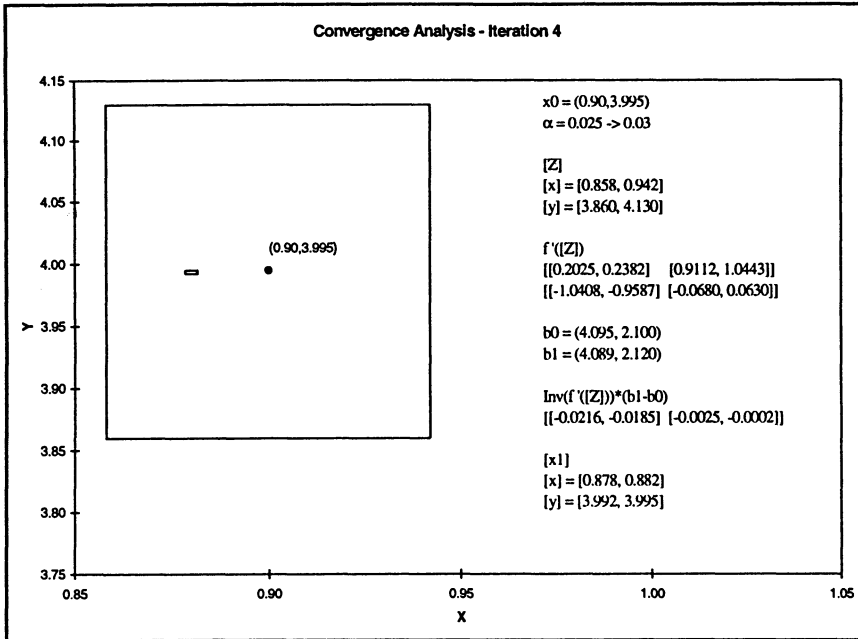


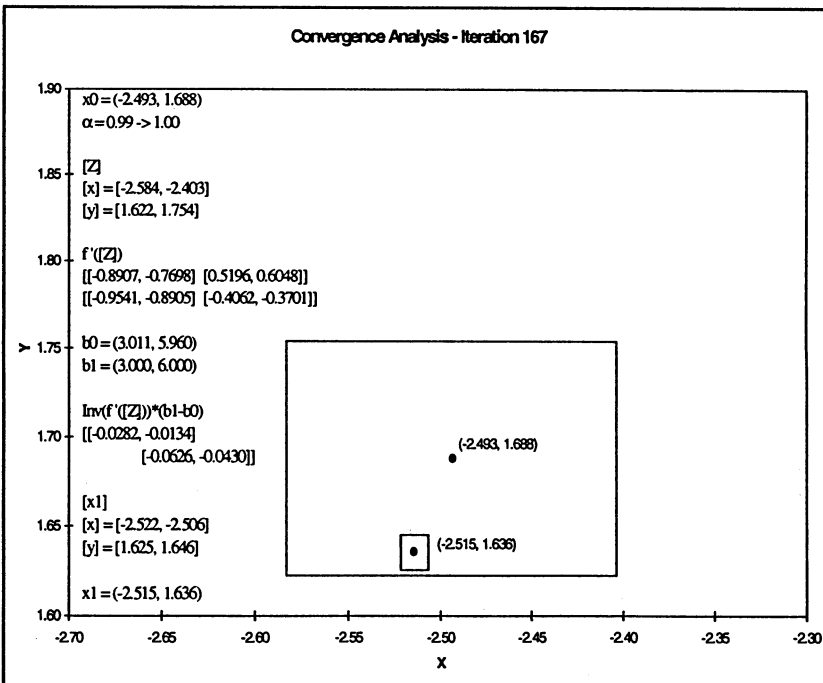
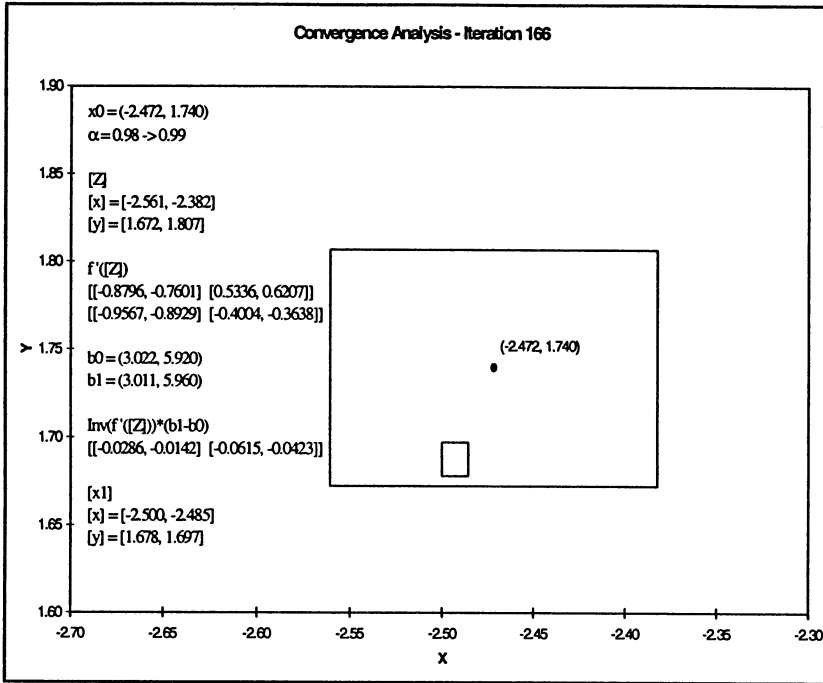
Figure 19.8 Solution Trajectory

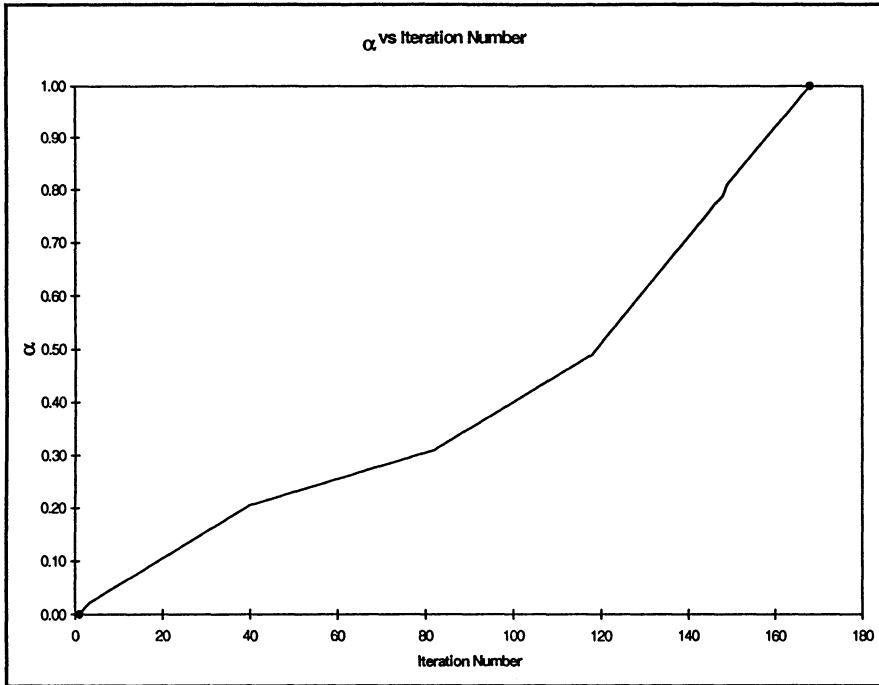
As the homotopy is followed, the following eight graphs visually describe the COAST algorithm for maintaining a consistent solution and the cumulative values of α .











19.5 LINE TANGENT TO TWO CIRCLES

We demonstrate here the situation of creating a line tangent to two circles. The unknown line has two dimensions: the angle, θ , and length, r , of a perpendicular vector from the origin to the line. Additionally, let there be two attributes: distance from the point at $(-3,3)$ and distance from the point at $(2,2)$. The two attributes are each equations in terms of the dimensions: $\sqrt{(3\sin\theta - 3\cos\theta - r)^2}$ and $\sqrt{(2\sin\theta + 2\cos\theta - r)^2}$, respectively. The user begins by initially constraining the line as follows:

$$\theta_0^C: \quad M_0: \quad \begin{cases} \text{dist}(-3,3) = 3 \\ \text{dist}(2,2) = 1 \end{cases} \rightarrow \begin{cases} r = 3.78 \\ \theta = 0.97 \end{cases}$$

The user then decides that the point should be distances 2 and 4 from $(-3,3)$ and $(2,2)$. The two attributes are solved simultaneously yielding the solution, $(5.55, 1.78)$:

$$\theta_0^C: \quad M_0: \\ \left\{ \begin{array}{l} \text{dist}(-3,3) = 2 \\ \text{dist}(2,2) = 4 \end{array} \right. \rightarrow \left\{ \begin{array}{l} r = 5.55 \\ \theta = 1.78 \end{array} \right.$$

The two distance attributes at the beginning of the problem can be interpreted as in Figure 19.9-A. When the constraints are modified, however, there are two possible solutions (see Figure 19.9-B).

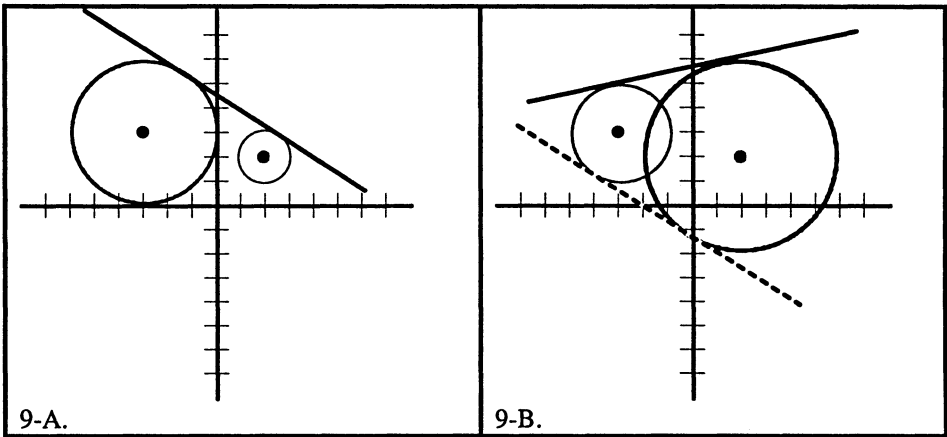


Figure 19.9 Simple Variational Design Example

As the radii of the circles are changed, the dimensions of the desired line create a trajectory from their original positions to their new positions (as shown in Figure 19.10).

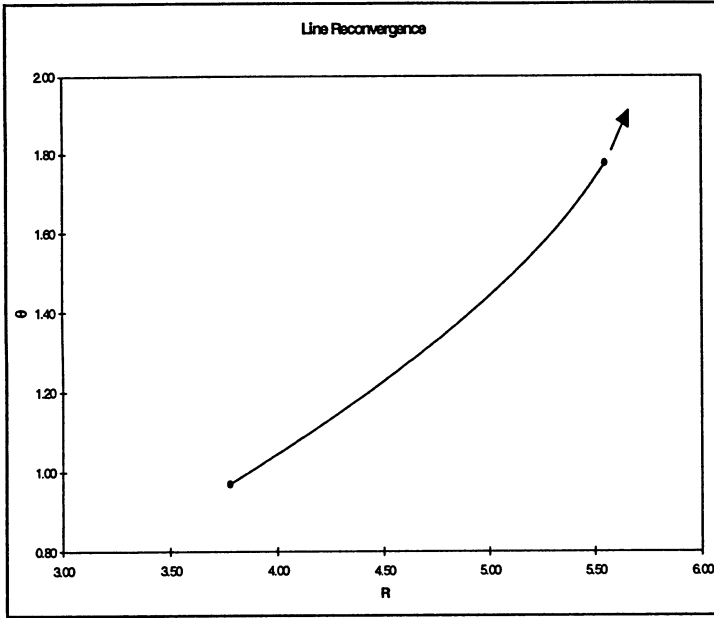
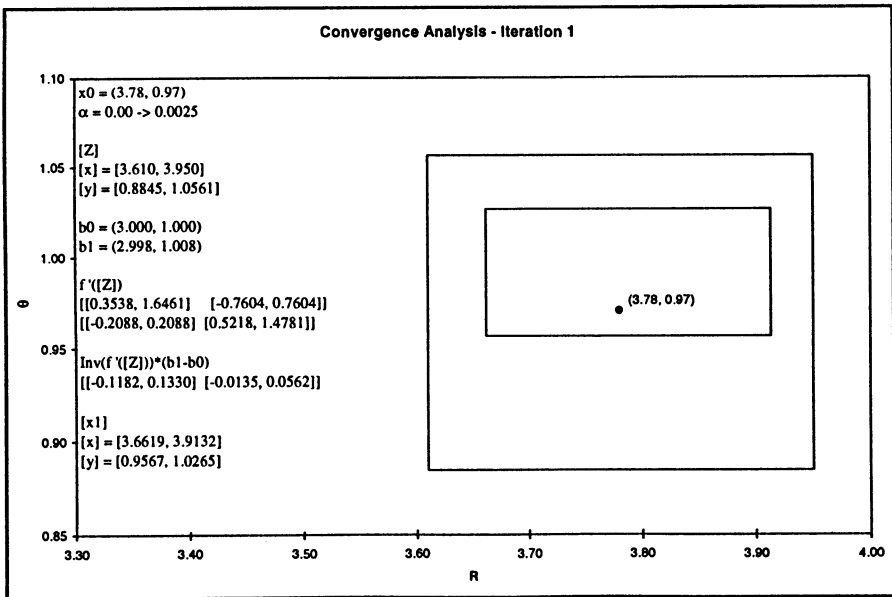
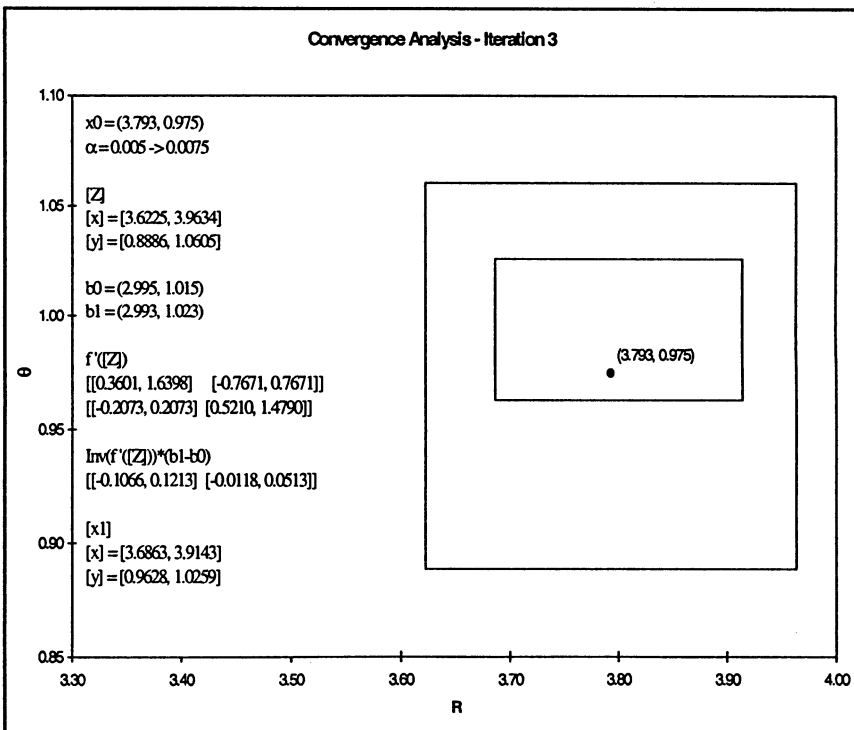
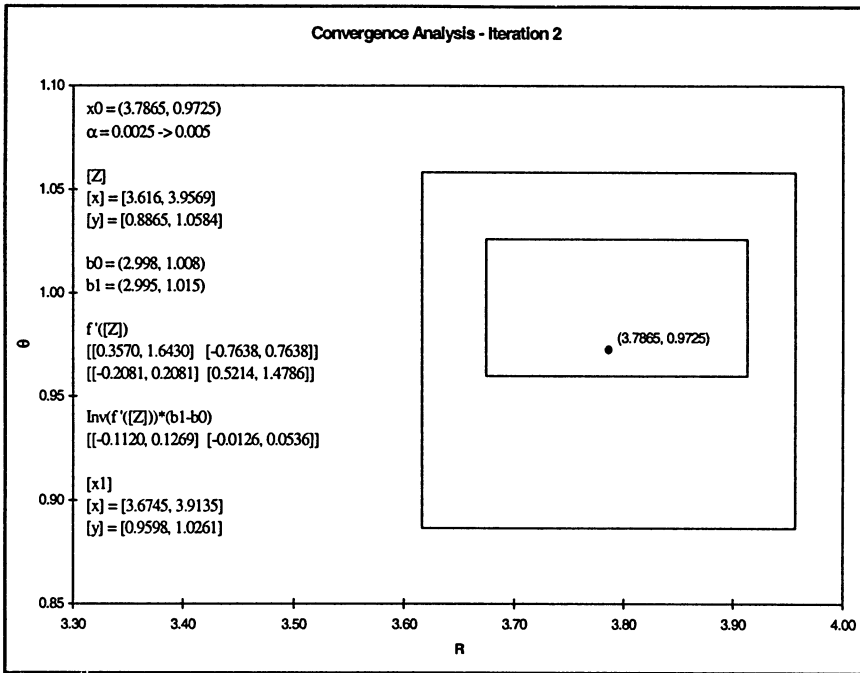
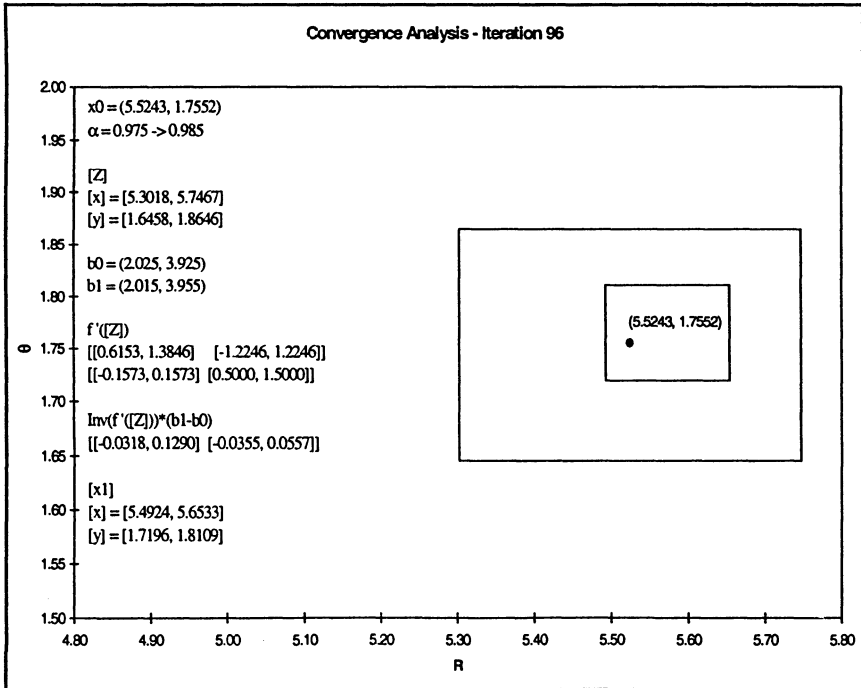
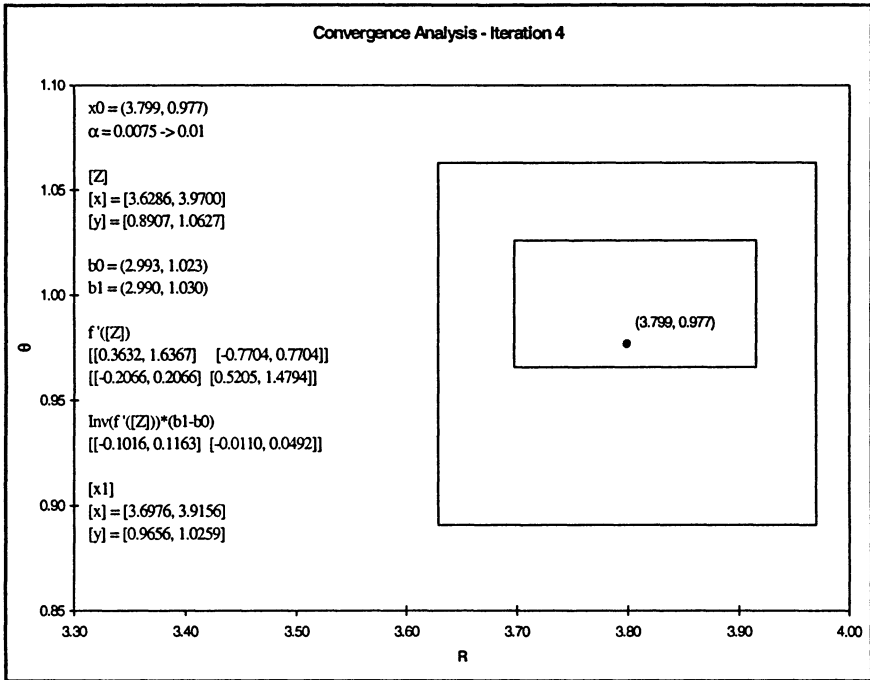


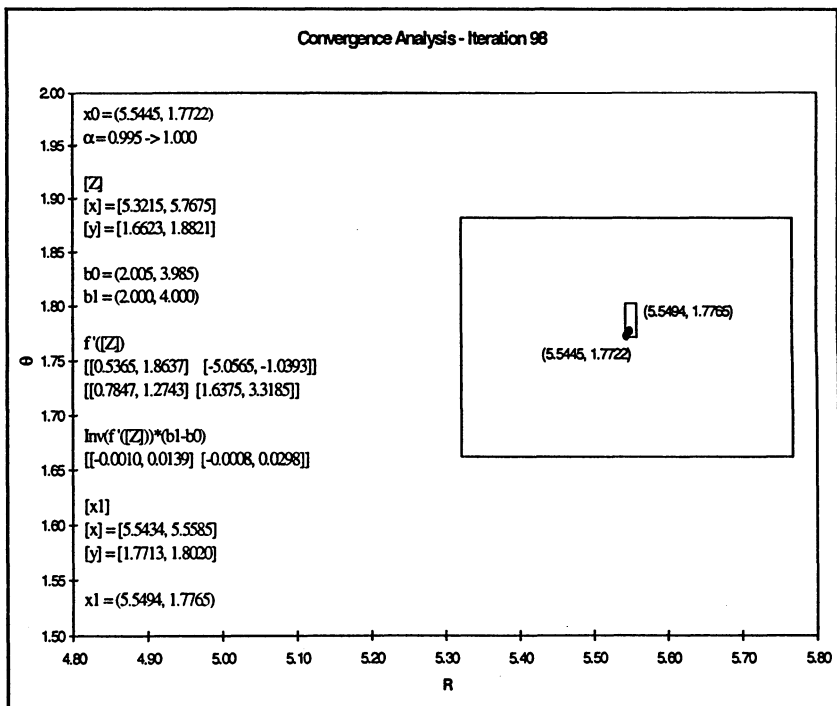
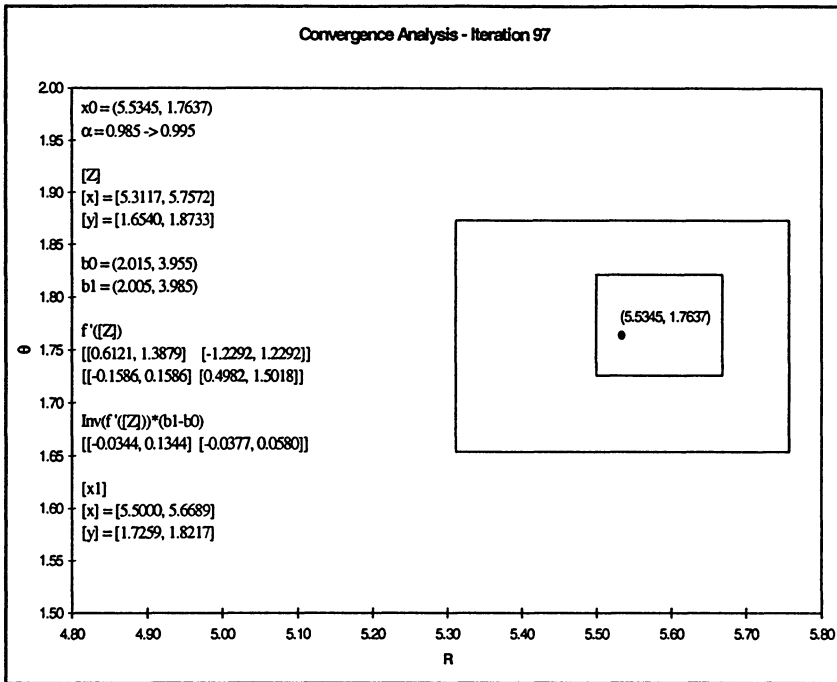
Figure 19.10 Solution Trajectory

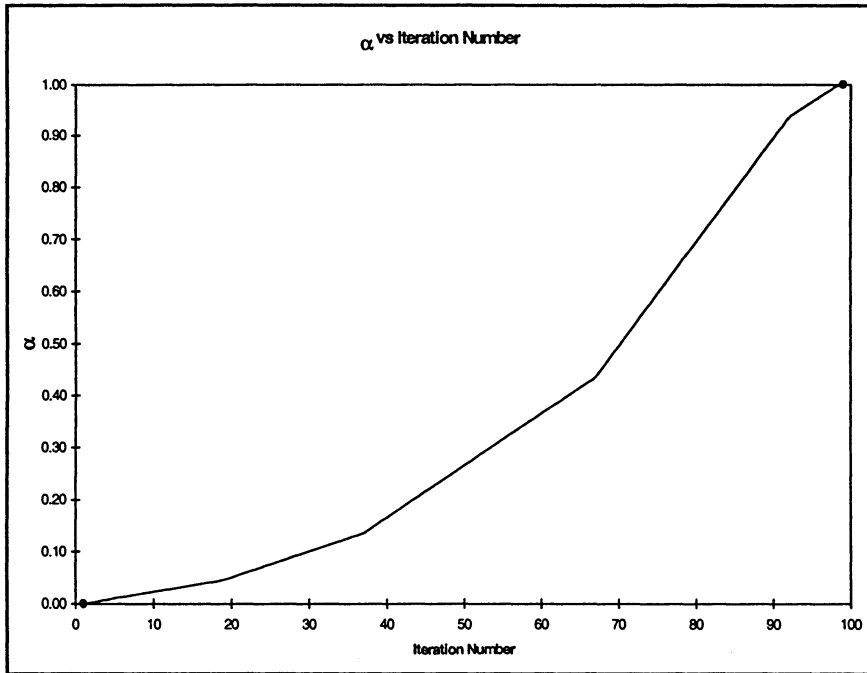
As the homotopy is followed, the following eight graphs visually describe the COAST algorithm for maintaining a consistent solution and the cumulative values of α .











19.6 HELICAL COMPRESSION SPRING (CONTINUED)

This section contains detailed information about the re-convergence in the helical compression spring example given in Section 19.2.

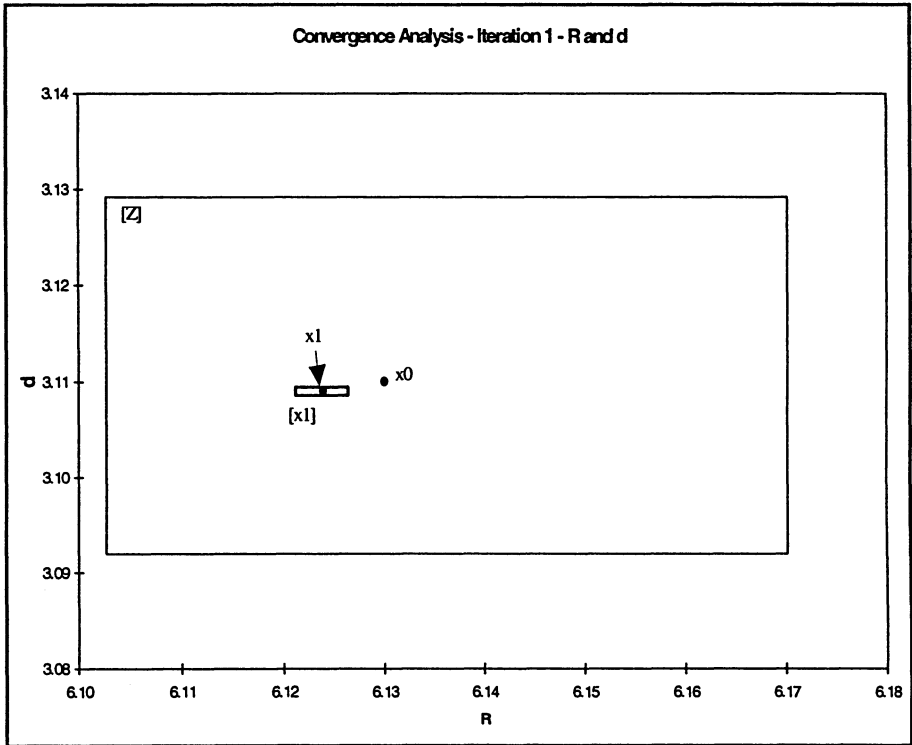
Beginning from M_0 (four possible solutions - first one is the desired one):

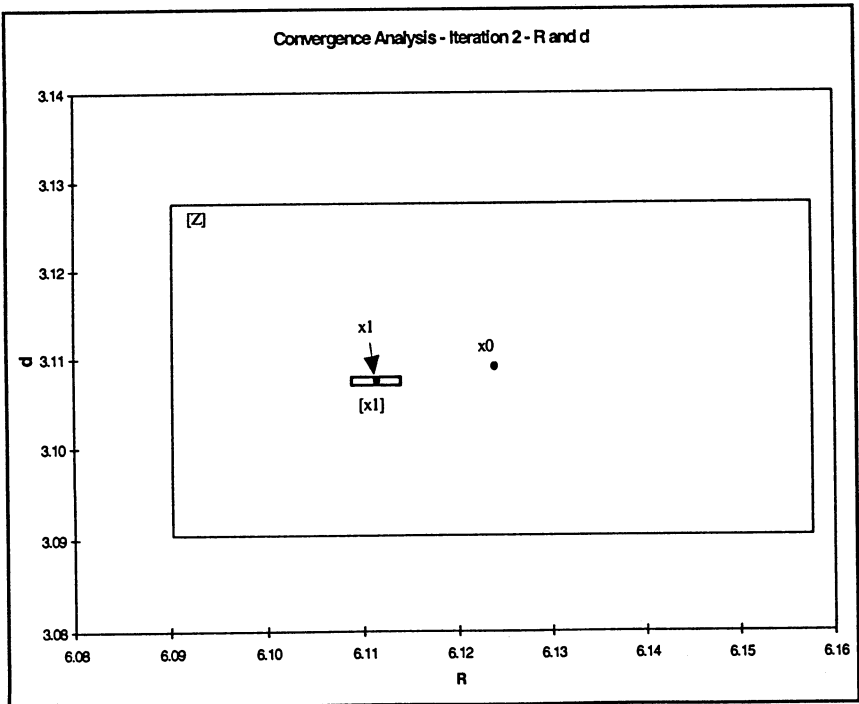
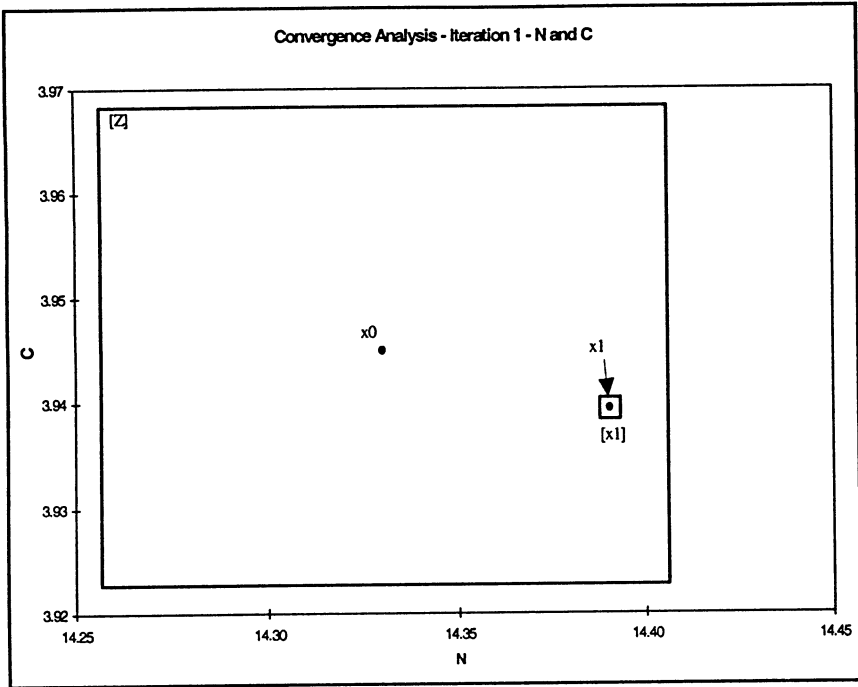
1. $(R, d, N, C) = (6.13, 3.11, 14.33, 3.945)$
2. $(R, d, N, C) = (-5.36, -3.01, -18.86, 3.5618)$
3. $(R, d, N, C) = (0.063, 0.204, 246.4, 0.6169)$
4. $(R, d, N, C) = (-0.0064, -0.206, -248.5, 0.6168)$

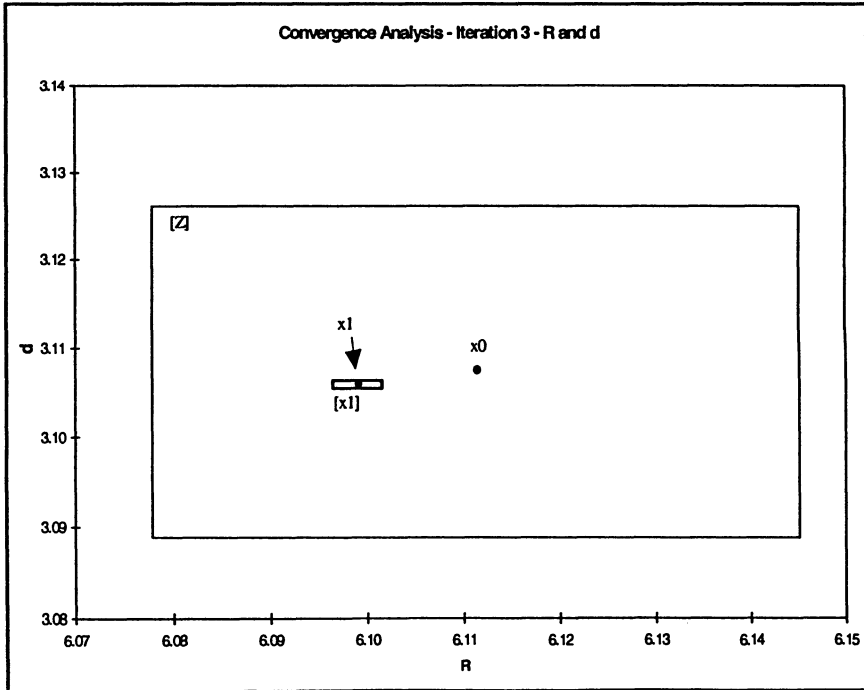
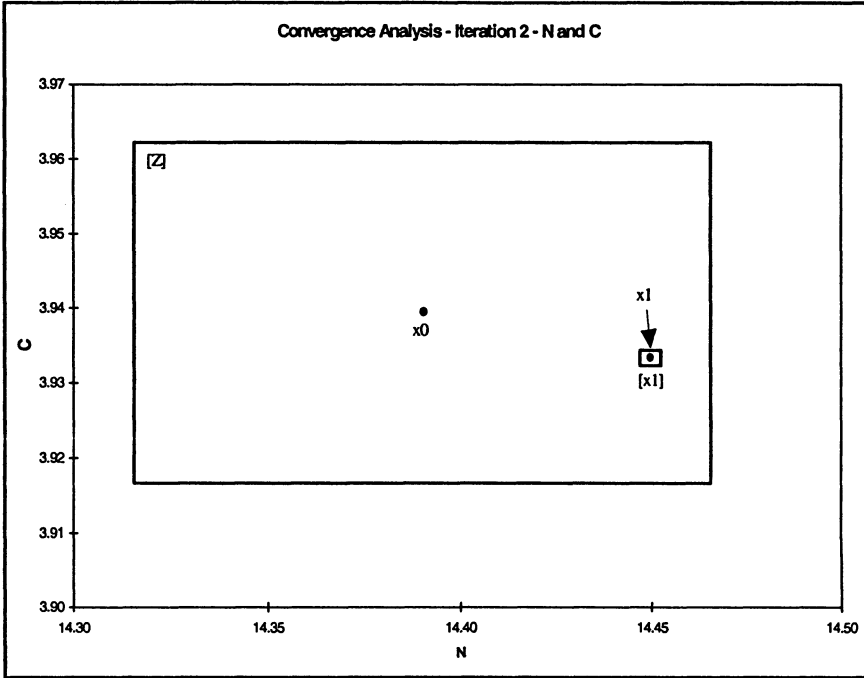
Now create a helical compression spring according to the updated specifications:

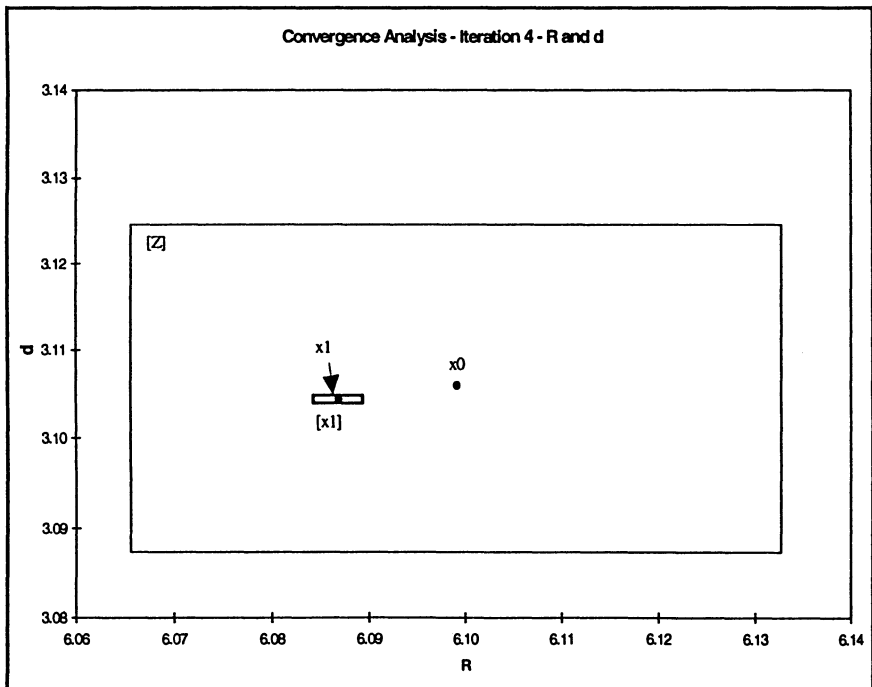
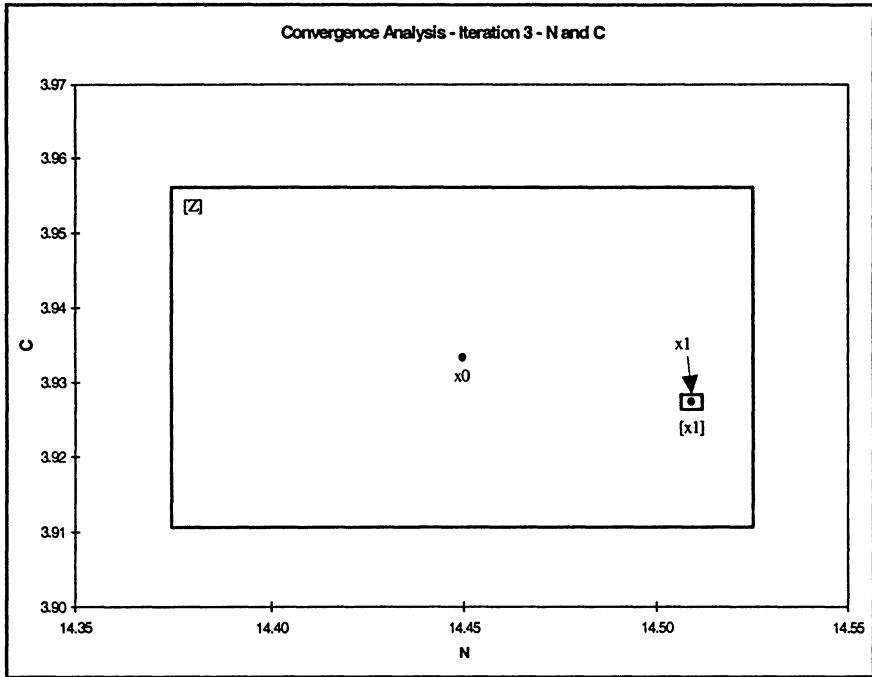
1. $\sigma_{all} = 1.303 \times 10^3$ MPa
2. $G = 7.929 \times 10^4$ MPa
3. $F = 2$
4. $\delta = 12.7$ mm
5. $L_f = 127.0$ mm
6. $P = 444.82$ N

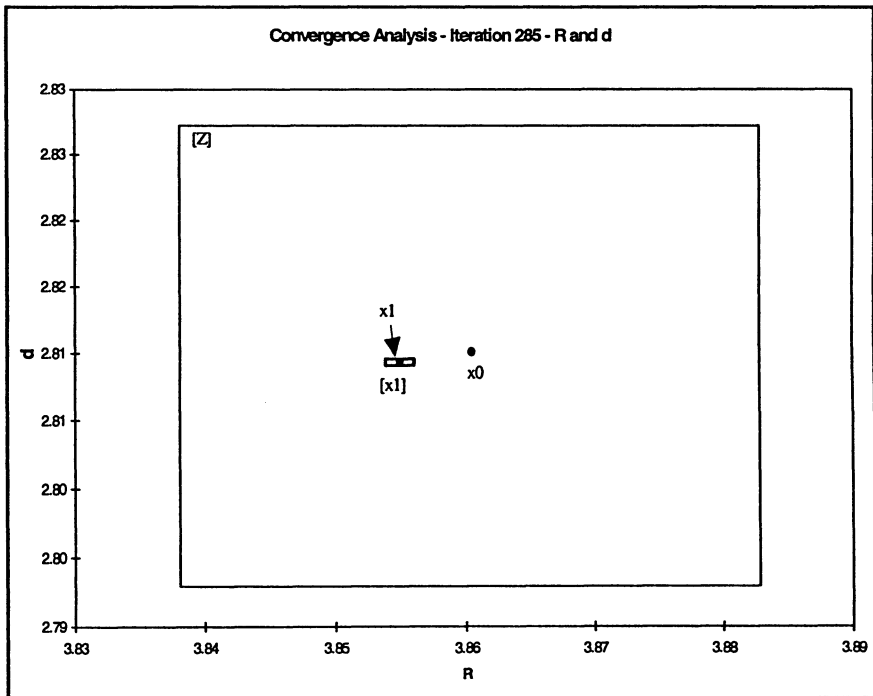
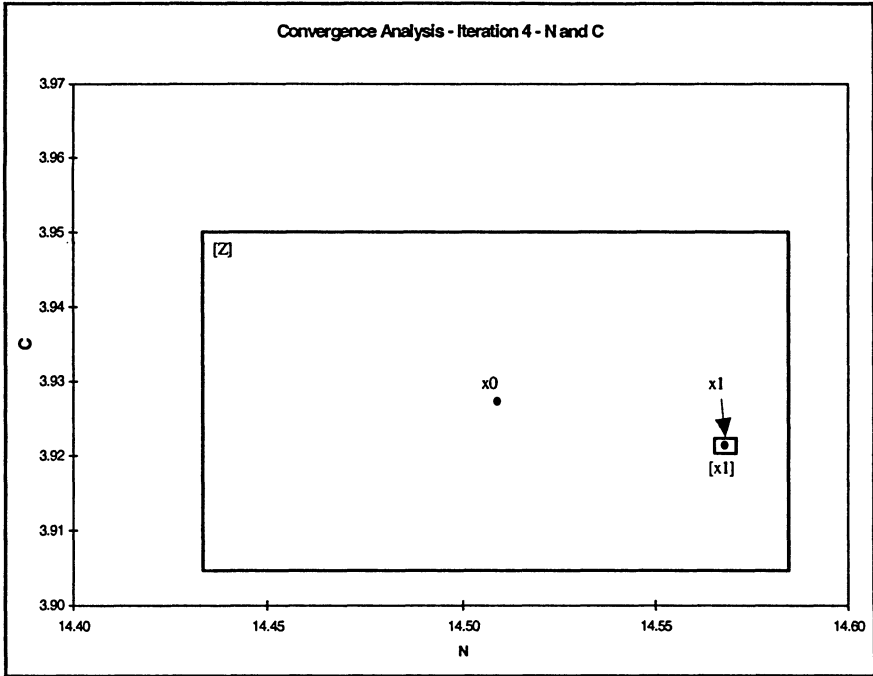
In the following graphs, the updated values of (R, d, N, C) are tracked from their original values (M_0) to the new desired solution, $M_1=(3.84, 2.81, 38.7, 2.74)$.

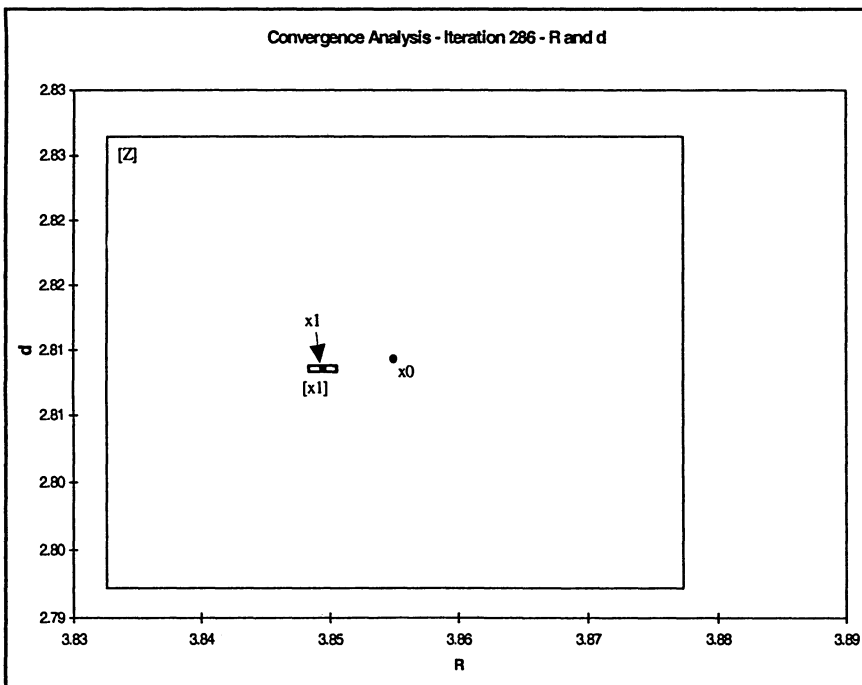
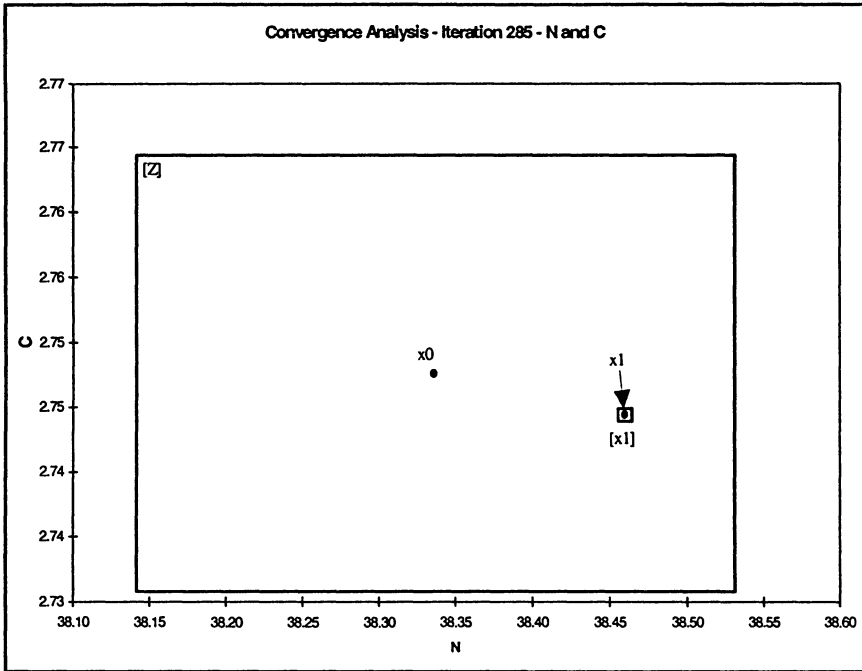


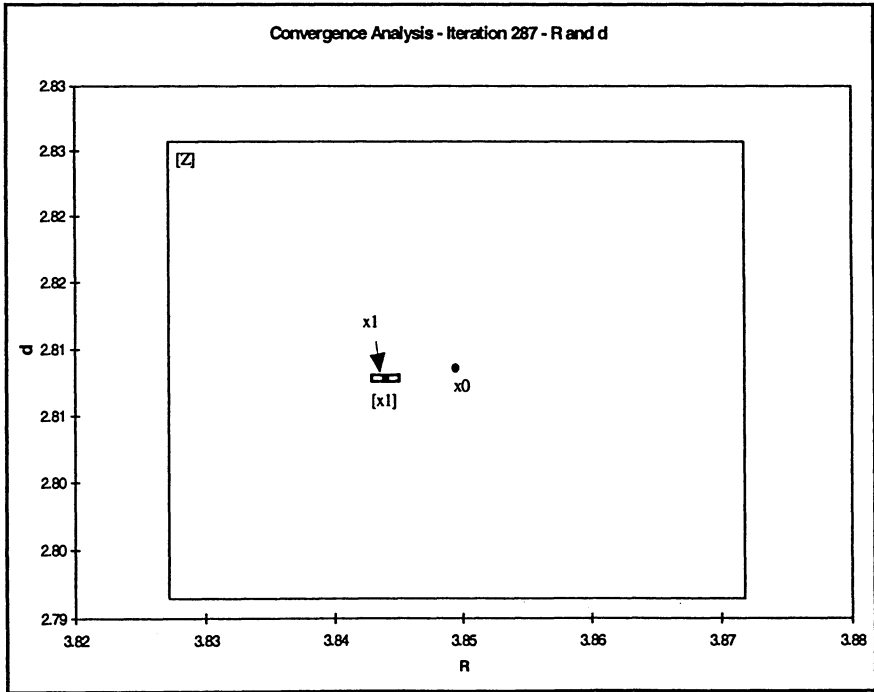
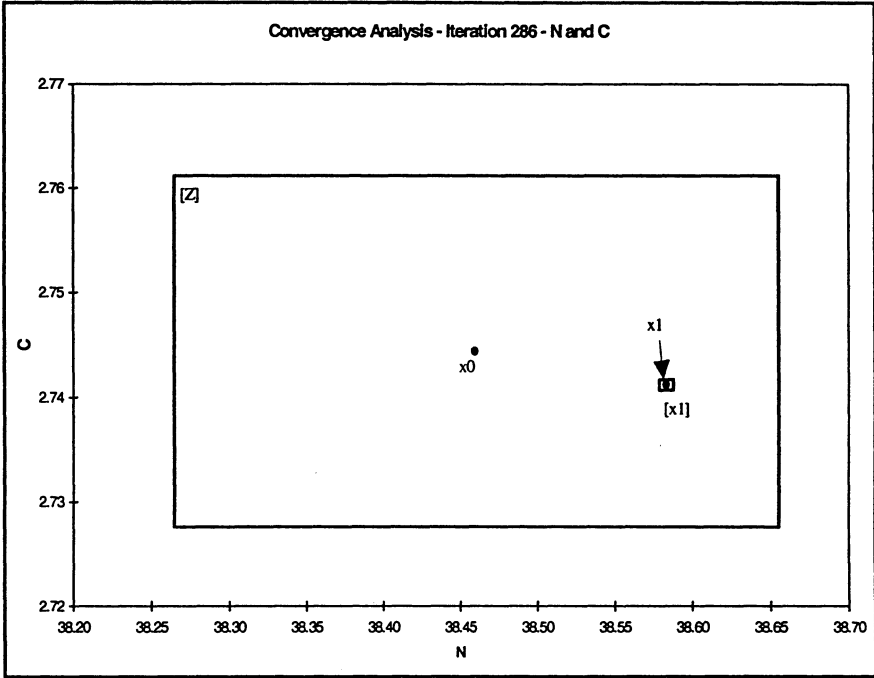


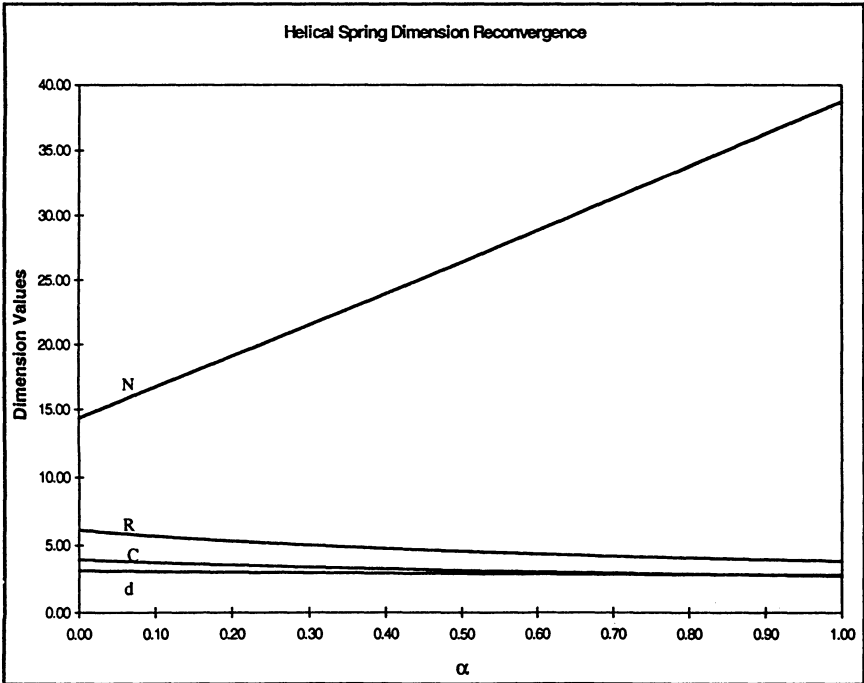
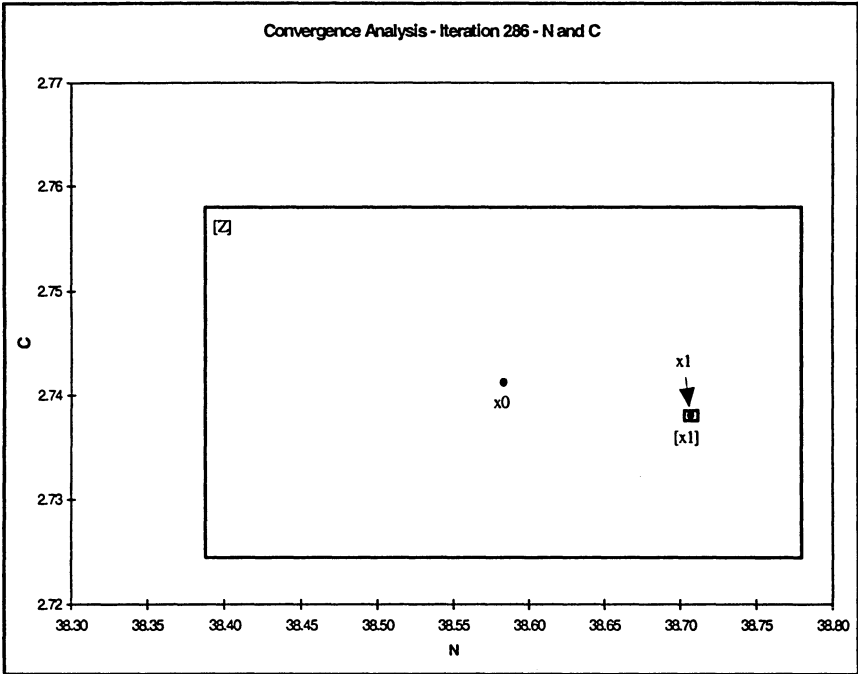


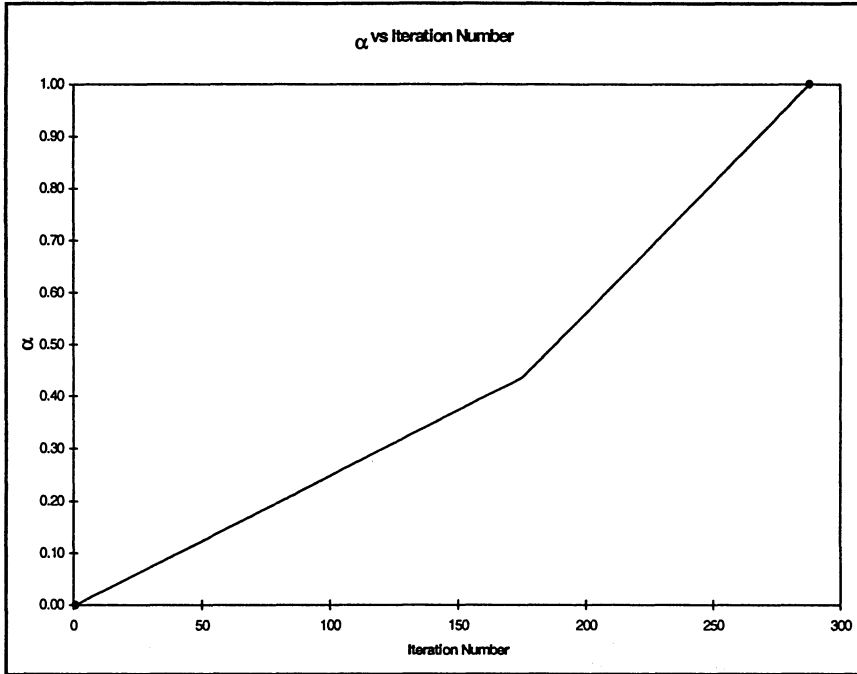












APPENDIX A - CONSTRAINT MODEL OF WORMGEAR

There are nine dimensions which describe a wormgear assembly:

N_i	=	Input Speed (RPM),	V	=	Lifting Speed (fpm)
d	=	Drum Diameter (in),	F	=	Lifting Load (lb)
η	=	Efficiency,	N_w	=	Number of threads
P	=	Pitch Angle (deg),	ϕ	=	Pressure Angle (deg)
λ	=	Lead Angle (deg),			

From those nine dimensions, numerous higher order attributes can be defined:

$$\text{Output Speed: } N_o = \frac{12V}{\pi d}, \quad \text{Reduction Ratio: } R = \frac{\pi d N_i}{12V}$$

$$\text{Output Power: } P_o = \frac{FV}{33000}, \quad \text{Input Power: } P_i = \frac{FV}{33000 \cdot \eta}$$

Worm Lead: $l = N_w P$, Worm Pitch Diameter: $D_w = \frac{N_w P \cot(\lambda)}{\pi}$

Wormgear Teeth Number: $N_g = \frac{\pi d N_i N_w}{12V}$

Worm Outside Diameter: $D_{wo} = 0.6366 P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi}$

Wormgear Pitch Diameter: $D_g = \frac{d P N_i N_w}{12V}$

Center Distance: $C = 0.5 \cdot \left(\frac{d P N_i N_w}{12V} + \frac{N_w P \cot(\lambda)}{\pi} \right)$

Wormgear Outside Diameter: $D_{go} = \frac{0.083333 d P N_i N_w}{V} + 0.6366 P \cos(\lambda)$

Length of Threaded Portion:

$$F_w = \sqrt{0.20263 P^2 + \frac{0.1061 d P^2 N_i N_w \cos(\lambda)}{V} + 0.20263 P^2 \cos(2\lambda)}$$

Wormgear Face Width:

$$F_g = 1.125 \cdot \sqrt{\left(0.6366 P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi} \right)^2 - \left(0.764 P \cos(\lambda) + \frac{N_w P \cot(\lambda)}{\pi} \right)^2}$$

Tangent Force: $F_t = \frac{3.81818 \pi F V}{N_i N_w P}$

Normal Force: $F_n = \frac{3.81818 \pi F V \csc(\lambda) \sec(\phi)}{N_i N_w P}$

Radial Force: $F_r = \frac{3.81818 \pi F V \csc(\lambda) \tan(\phi)}{N_i N_w P}$

Thrust Force: $F_{thrust} = \frac{3.81818 \pi F V \cot(\lambda)}{N_i N_w P}$

Wormgear Pitch Line Velocity: $V_{fg} = \frac{N_i N_w P}{12}$

Dynamic Load: $F_d = 0.000265152 F \pi + \frac{3.81818 \pi F V}{N_i N_w P}$

CHAPTER 20

CASES IN EVOLUTIONARY DESIGN PROCESSES

In Chapter 10, a design search algorithm was developed according to the evolutionary transformation process model presented in Chapter 6. The Design Search Algorithm is based on a set of production rules, and an AND/OR tree representation is used to search for a consistent (i.e., physically realizable) design solution. A prototype system, called CADAT, has been developed to implement the design decomposition framework. In this chapter, we illustrate the application of the Design Search Algorithm to several nontrivial, “real world” design problems.

20.1 AUTOMOBILE DESIGN EXAMPLE

The automobile is a self-propelled, four-wheeled, steerable vehicle for transporting people on land. All passenger cars, trucks, and buses have certain things in common: (1) the power plant, or *engine*; (2) the *chassis*, which supports the engine and wheels and includes the *frame* and the *steering* and *brake* systems; (3) the *power train*, which transmits the power from the engine to the car wheels; and (4) the *body*. Technical and operational details related to the design of the main parts of automobiles and their components are provided in Appendix A. This section uses the car domain as a representative design domain to test the Design Search Algorithm described in Chapter 10.

20.1.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES

The Design Description Properties

The design description properties (structural attributes) specify the configuration of actual cars:

- | | |
|----------------------------|-------------------------------|
| (m_1) 4-wheel drive | (m_4) absorbent front end |
| (m_2) 4-wheel steering | (m_5) air bag |
| (m_3) 6-8 cylinders | (m_6) air cooled engine |

- | | |
|---|---|
| (m_7) air deflector | (m_{25}) hydraulic disk brakes |
| (m_8) an engine that
deflects down | (m_{26}) large pistons and
cylinders |
| (m_9) anti-lock braking
system (ABS) | (m_{27}) light weight |
| (m_{10}) automatic belts | (m_{28}) liquid cooling system |
| (m_{11}) catalytic converter | (m_{29}) low and small
structure |
| (m_{12}) deep thread patterns | (m_{30}) muffler |
| (m_{13}) disconnecting fan
system | (m_{31}) power brakes |
| (m_{14}) drum brakes | (m_{32}) powerful starting
engine |
| (m_{15}) electric-powered | (m_{33}) radial tire |
| (m_{16}) electronic ignition | (m_{34}) richer mixture fuel |
| (m_{17}) extra differential | (m_{35}) rigid passenger
compartment |
| (m_{18}) extra strong door | (m_{36}) stabilizers in the
front |
| (m_{19}) extra strong roof | (m_{37}) suspension system |
| (m_{20}) fog lights | (m_{38}) tubeless tire |
| (m_{21}) fuel injection | (m_{39}) windshield defroster |
| (m_{22}) high ground clearance | (m_{40}) windshield washer and
wiper |
| (m_{23}) high transmission ratio | |
| (m_{24}) horn | |

The Specification Properties

The specification properties (functional attributes) describe the requirements and

constraints:

- | | |
|---|--|
| (r_1) AERODYNAMIC DESIGN | (r_{16}) PASSABLE CAR |
| (r_2) DIESEL ENGINE | (r_{17}) RELIABLE TIRE |
| (r_3) DOESN'T POLLUTE THE
ATMOSPHERE | (r_{18}) SAFE CAR |
| (r_4) EASY PARKING | (r_{19}) SAFE IN ACCIDENTS |
| (r_5) ECONOMIC ENGINE | (r_{20}) SAFE IN BAD WEATHER |
| (r_6) ECONOMICAL | (r_{21}) SAFE IN FLIPPING OVER |
| (r_7) GOOD BRAKES | (r_{22}) SAFE IN HEAD-ON
COLLISIONS |
| (r_8) HEAVY CAR | (r_{23}) SAFE IN HIGH SPEED
DRIVING |
| (r_9) HIGH POWER OUTPUT | (r_{24}) SAFE IN OFF-HIGHWAY
ROAD |
| (r_{10}) HIGH SPEED | (r_{25}) SAFE IN POOR EXTERNAL
CONDITIONS |
| (r_{11}) HIGH VOLUME OF THE
COMBUSTION CHAMBER | (r_{26}) SAFE IN POOR VISIBILITY |
| (r_{12}) LOW FUEL CONSUMPTION | (r_{27}) SAFE IN SIDE COLLISIONS |
| (r_{13}) LOW MAINTENANCE COSTS | (r_{28}) SMALL CAR |
| (r_{14}) MECHANICALLY DEPENDABLE
and DURABLE | (r_{29}) SMALL ENGINE |
| (r_{15}) OFF-HIGHWAY TIRE | (r_{30}) STRONG ENGINE |

20.1.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the car design domain is expressed in terms of the rules presented in this section. The set of rules is held in *rule memory*; and a *working memory* holds or represents the current state of the process, which is (in general) a conjunction of structural attributes and the

(remaining) current specifications. These rules rely very heavily on the domain-specific knowledge presented in Appendix A.

(RULE 1) PASSABLE

If the car has OFF-HIGHWAY TIRE & 4-wheel drive & extra differential & high ground clearance & light weight
Then the car is PASSABLE

Cause: (1) OFF-HIGHWAY TIRE - off-highway tires and snow tires have deeper treads or separate cleats that bite through snow, slush, or dirt to grip the firmer surface beneath; (2) 4-wheel drive & extra differential - many vehicles now have full or part-time four-wheel-drive (4WD). Part-time 4WD cars are driven in 2WD on paved roads. Most modern 4WD cars add an extra differential between the front and rear wheels, so the front and rear driving wheels can turn at minutely differing rates when driven on pavement, to avoid drive train damage. In some vehicles with full-time 4WD, limited-slip differentials couple the front and rear final drive gears. Each driving axle has its own differential as well. These differentials allow front and rear wheels to turn at slightly varying rates to compensate for minor differences in tire diameters or drive-gear ratios. This slippage allows the vehicle with full-time 4WD to run on paved roads without damage to its drive train. This also provides a better passable car; (3) high ground clearance - a high ground clearance is needed in order to avoid colliding in objects on the surface of the road (e.g., rocks) - this attribute provides a better passable car; (4) light weight - if the car weight is low and the road is soft (due to mud or snow) the car will not sink - a greater pass-ability.

(RULE 2) SAFETY-1

If the car is SAFE IN POOR EXTERNAL CONDITIONS
Then the car is SAFE

(RULE 3) SAFETY-2

If the car is SAFE IN HIGH SPEED DRIVING
Then the car is SAFE

(RULE 4) SAFETY-3

If the car is SAFE IN ACCIDENTS
Then the car is SAFE

(RULE 5) SAFETY IN POOR EXTERNAL CONDITIONS

If the car is SAFE IN POOR VISIBILITY & SAFE IN BAD

WEATHER & SAFE IN OFF-HIGHWAY ROAD

Then the car is SAFE IN POOR EXTERNAL CONDITIONS

(RULE 6) SAFETY IN ACCIDENTS

If the car is SAFE IN HEAD-ON COLLISIONS & SAFE IN SIDE COLLISIONS & SAFE IN FLIPPING OVER & has automatic belts

Then the car is SAFE IN ACCIDENTS

Cause: a seat belt is a strap -- usually a shoulder harness -- that restrains an occupant in the seat, preventing him or her from being thrown out of the seat during a sudden stop or change in direction. Fewer than 20 percent of automobile occupants routinely use safety belts, even though convincing evidence exists as to their value, and legislation has been passed that requires seat belts in all cars. Because of the poor response of the driving public to devices that require their active participation, safety researchers have developed automatic, or passive, restraint systems, which protect occupants without any action on their part. Two basic types of passive restraints have been produced: (1) the automatic belt, which fastens around the occupant when the car door is closed (particularly important for safety in accidents); and (2) the air bag.

(RULE 7) SAFETY IN HIGH SPEED DRIVING

If the car has GOOD BRAKES & 4-wheel steering (4WS) & stabilizers in the front suspension system & extra strong roof & rigid passenger compartment

Then the car is SAFE IN HIGH SPEED DRIVING

Cause: (1) 4-wheel steering (4WS) - if the car has 4-WHEEL STEERING, high-speed maneuvers are safer because "fishtailing" is reduced; (2) stabilizers - a stabilizer is a long steel rod that is attached at each end to the two lower control arms. Its purpose is to prevent excessive lean-out on turns. Thus, it is especially important in high speed driving; (3) extra strong roof - an extra strong roof protects the passengers when the car is flipping and falls on the roof. It is especially important for high-speed cars; (4) rigid passenger compartment - when the passengers have to be protected in a car that moves in a high speed, and might be flipped over, a rigid passenger compartment may be used.

(RULE 8) SAFETY IN POOR VISIBILITY

If the car has fog lights & windshield washer and wiper & horn & windshield defroster

Then the car is SAFE IN POOR VISIBILITY

Cause: In order to improve sensing (in poor visibility conditions), the following attributes might be used: (1) fog lights - prevent reflecting of the car lights; (2) windshield washer and wiper & defroster - to make sure that the windshield is clear; and (3) horn - to warn other drivers.

(RULE 9) SAFETY IN BAD WEATHER

If the car has GOOD BRAKES & deep thread patterns & anti-lock braking system (ABS)
Then the car is SAFE IN BAD WEATHER

Cause: (1) deep thread patterns are especially important when the road is wet. The forward portion of a tire's contact patch wipes away water so that the rest of the patch grips a drier surface. Continuous channels from the center to the edge of the tread direct the water outward. Without a carefully designed tread, water would form a wedge and cause the tire to lift off the road. This so-called aquaplaning phenomenon is one reason that smooth tires (whether they are intentionally smooth racing slicks or regular tires that have been worn bald) are dangerous in bad conditions. Tread patterns are especially important when the road is wet (for a off-highway tire); (2) the anti-lock braking system prevent the lock of the wheel because of the constant pressure the driver applies in case of sudden stop, and a computer is controlling the braking- it's main purpose is to prevent sliding especially in bad weather.

(RULE 10) SAFETY IN OFF-HIGHWAY ROAD

If the car has OFF HIGHWAY TIRE & stabilizers in the front suspension system
Then the car is SAFE IN OFF HIGHWAY ROAD

(RULE 11) SAFETY IN HEAD-ON COLLISIONS

If the car has absorbent front end & air bag & an engine that deflects down
then the car is SAFE IN HEAD-ON COLLISIONS

Cause: (1) absorbent front-end - when an accident occurs the front-end is absorbing the force ,instead of passing it to the passengers in the car; (2) air bag - in a head-on collision two air bags -- one in the steering column and one in the right side of the dashboard -- pop out and instantly inflate, forming cushions that prevent the occupants from striking hard surfaces, such as the dashboard or windshield; (3) deflecting down engine - if an accident (in head-on collisions) occurs, the motor is deflected downwards and so not colliding into the passengers inside the car.

(RULE 12) SAFETY IN SIDE COLLISIONS

If the car has extra strong door
Then the car is SAFE IN SIDE COLLISIONS

Cause: extra strong doors protects the passengers from objects that collide from the side.

(RULE 13) SAFETY IN FLIPPING OVER

If the car has extra strong roof & rigid passenger compartment
Then the car is SAFE IN FLIPPING OVER

Cause: an extra strong roof protects the passengers when the car is flipping and falls on the roof.

(RULE 14) EASY PARKING

If the car has 4-wheel steering
Then the car is EASY PARKING

(RULE 15) HIGH-SPEED

If the car has AERODYNAMIC DESIGN & STRONG ENGINE & high transmission ratio & light weight
Then the car has HIGH SPEED

Cause: (1) AERODYNAMIC DESIGN - designers of high-speed cars must also take into account aerodynamic concepts such as the boundary layer. This is the layer of air nearest the skin of the car where the effects of the turbulence caused by air resistance are exhibited most strongly. It is desirable to keep turbulence to a minimum, so cars are designed to keep the stream of air flowing around the car as undisturbed as possible - hence the term streamlining; (2) STRONG ENGINE - cars that need to move in high-speed must use a strong engine, because it is the part that supplies the kinetic energy that moves the car; (3) high transmission ratio - the transmission ratio determines the ratio between power and speed. Thus, to enable the car to move in high-speed, an option for a high transmission ratio may be considered; (4) light weight - if the car weight is low, then for the same engine, the car with a light weight can move in high speed.

(RULE 16) AERODYNAMIC DESIGN

If the car has low and small structure & air deflector
Then the car has an AERODYNAMIC DESIGN

Cause: drag can be dramatically reduced by an air deflector. This device, which comes in various shapes, improves the streamlining by deflecting air around the car. Carefully designed air deflectors improves dramatically the aerodynamic shape of the car. An air deflector is also a very cheap device.

(RULE 17) STRONG ENGINE

If the car has HIGH VOLUME OF THE COMBUSTION CHAMBER & DIESEL ENGINE & richer mixture fuel & muffler
Then the car has a STRONG ENGINE

Cause: (1) HIGH VOLUME OF THE COMBUSTION CHAMBER - a higher combustion chamber volume gives better power capacity, thus, a strong engine; (2) muffler - larger and stronger engines are noisier and require a muffler or sound deadener; usually a canister with an inner shell that dissipates sound wave energy within the muffler before exhaust gases are permitted to escape.

(RULE 18) HIGH VOLUME OF THE COMBUSTION CHAMBER

If the car has large pistons and cylinders & 6-8 cylinders
Then the car has a HIGH VOLUME OF THE COMBUSTION CHAMBER

Cause: the volume of the combustion chamber can be increased by increasing the size of the piston and cylinder and by increasing the number of cylinders.

(RULE 19) OFF-HIGHWAY TIRE

If the car has RELIABLE TIRES & the tire has deep thread patterns
Then the car has OFF-HIGHWAY TIRE

Cause: (1) RELIABLE TIRES - an off-highway tire has to be a reliable tire because of the especially hard conditions; (2) deep thread patterns - for a off-highway tire, tread patterns are especially important when the road is wet.

(RULE 20) ECONOMICAL

If the car has LOW FUEL CONSUMPTION
Then the car is ECONOMICAL

(RULE 21) LOW FUEL CONSUMPTION

If the car has AERODYNAMIC DESIGN & ECONOMIC ENGINE & DIESEL ENGINE & disconnecting fan system & light weight

Then the car has LOW FUEL CONSUMPTION

Cause: (1) DIESEL ENGINE - a diesel engine may be selected because of its operating advantages, such as low maintenance costs, greater efficiency, high power output, and fuel economy under all loads. It is also a very strong energy source; (2) disconnecting fan system - demand-actuated fan drive systems turn the fan on and off as needed, either by a temperature-sensing device or by a centrifugal clutch that disconnects the fan when the engine, and hence vehicle, speed is high enough to provide adequate cooling. Having the fan disconnected when not needed can save a considerable amount of energy, thus produces a low fuel consumption; (3) light weight - light weight cars can be created by the use of plastics and lightweight materials, and result with low fuel consumption.

(RULE 22) RELIABLE TIRE

If the car has tubeless tire & radial tire
 Then the car has RELIABLE TIRE

Cause: (1) tubeless tire - since a tubeless tire has no tube the user doesn't have to deal with the problem of "punctures" - it is considered a reliable tire; (2) radial tire - a radial tire has reinforcing belts under its tread, but radial belt cords are angled closer to the tire's center line. The lack of bias side-wall reinforcement makes a radial's side-walls more flexible. This gives the tread a better grip and longer life, and therefor it's considered a reliable tire.

(RULE 23) ECONOMIC ENGINE

If the car has electronic ignition & fuel injection
 Then the car has ECONOMIC ENGINE

Cause: (1) electronic ignition - electronic ignition system with a fuel injection system (controlled by a computer) makes the engine more efficient and economic; (2) fuel injection - carburetors use various means to ensure an optimal mixture of gasoline and air under differing conditions, including idling and rapid acceleration. Instead of having a carburetor, an engine can have a system of fuel injection, which delivers a metered quantity of gasoline directly to each cylinder. Fuel injection has always been used with diesel engines; it has also been gaining in use with gasoline engines - and it's more economic then the usual systems.

(RULE 24) GOOD BRAKES-1

If the car has drum brakes
 Then the car has GOOD BRAKES

Cause: drum brakes are simple and cheap, and they fit for cars that have light weight, and are comparatively slow.

(RULE 25) GOOD BRAKES-2

If the car has hydraulic disk brakes
Then the car has GOOD BRAKES

Cause: hydraulic disk brakes (which most modern cars use) solve two main problems: (1) it is difficult to brake all the wheels equally; (2) the increased weight and speed of vehicles requires that the driver exert a greater pedal pressure.

(RULE 26) GOOD BRAKES-3

If the car has power brakes
Then the car has GOOD BRAKES

Cause: power brakes are used for very heavy or very fast cars. As vehicles became heavier and faster the pedal pressure required to brake the vehicle increased beyond a comfortable, safe level. Power brakes were developed to solve this problem. In automobiles they use the vacuum created by the engine during its intake stroke to increase the pressure applied to the piston in the master cylinder, reducing the required pedal pressure. If the power-assisting mechanism fails, or the engine stalls then the brakes will not fail completely (although greater pedal pressure will be needed).

(RULE 27) LOW MAINTENANCE COSTS-1

If the car has DIESEL ENGINE
Then the car has LOW MAINTENANCE COSTS

(RULE 28) LOW MAINTENANCE COSTS-2

If the car has air cooled engine
Then the car has LOW MAINTENANCE COSTS

Cause: an air cooled system is popularly used to power small cars or light weight cars, often requires no moving parts, and therefore requires low or no maintenance.

(RULE 29) HIGH POWER OUTPUT

If the car has DIESEL ENGINE
Then the car has HIGH POWER OUTPUT

(RULE 30) DIESEL ENGINE

If the car has powerful starting engine & liquid cooling
 system & muffler
Then the car uses a DIESEL ENGINE

Cause: (1) powerful starting engine - because of the unusually high compression ratios, diesel engines need a powerful starting system. Some diesel engines use an electric motor or an auxiliary gasoline engine, whereas others use compressed air or spark ignition to start the engine; (2) liquid cooling system - most modern automobiles, with the strong engines (especially diesel engine) have a liquid cooling system, which is far more efficient than other cooling systems.

(RULE 31) HEAVY

If the car has STRONG ENGINE
Then the car is HEAVY

Cause: heavy cars need high kinetic energy in order to move. A strong engine is the part that supplies the high kinetic energy.

(RULE 32) SMALL CAR

If the car has SMALL ENGINE
Then the car is SMALL

(RULE 33) SMALL ENGINE

If the car has air cooled engine
Then the car has SMALL ENGINE

(RULE 34) MECHANICALLY DEPENDABLE & DURABLE

If the car is electric-powered
Then the car is MECHANICALLY DEPENDABLE & DURABLE

Cause: an electric-powered car is mechanically more dependable and durable than is the gasoline-powered car.

(RULE 35) DOESN'T POLLUTE THE ATMOSPHERE-1

If the car has catalytic converter
Then the car DOESN'T POLLUTE THE ATMOSPHERE

Cause: A catalytic converter is a device in the exhaust system of an automotive engine that converts environmentally harmful exhaust gases into harmless gases by promoting a chemical reaction between a catalyst and the pollutants.

(RULE 36) DOESN'T POLLUTE THE ATMOSPHERE-2

If the car is electric-powered
Then the car DOESN'T POLLUTE THE ATMOSPHERE

20.1.3 CAR SYNTHESIS USING THE DESIGN SEARCH ALGORITHM (SEE CHAPTER 10.3)

Assume that the designer is faced with the problem of designing a car description that is able to achieve the following specifications:

1. The car is safe (r_{18});
2. The car is capable of high speed (r_{10});
3. The car has low fuel consumption (r_{12}).

The following constraints are further assumed: (1) the external conditions are good, (2) the maximum speed is 160 Km/h, and (3) the weight of the car is between 1 and 3 ton. These constraints will be useful in case the interpreter (or inference engine) identifies (matches) several production rules the condition parts of which satisfy the state of the working memory. In this case, the interpreter selects a preferred rule (that best matches the constraints) from the conflict set to fire (execute).

The designer (or design system) must search through the problem space for a pathway from the initial specifications to some state of the car description such that the specifications r_{18} , r_{10} , and r_{12} are achieved. Recall from chapter 10 that among the heuristics used to effect such a search is a strategy called *backward chaining* in which the designer in which the search process begins with the goal state (initial specifications). The preconditions to the attainment of the initial specifications are identified and these become sub-specifications. The search process is then applied recursively on the sub-specifications until eventually a set of structural attributes that correspond to the sub-specifications are identified. The Design Search Algorithm introduced in Section 10.3 is an instance of backward chaining. In attempting to solve our goal the Design Search Algorithm is served as the problem solving strategy. In the context of production systems this strategy will be embedded in the inference engine. In addition, the CADAT architecture (CASE-based Design Advisory Tool, see Section 10.4) is utilized in order to assist the problem of organizing and representing knowledge in the knowledge base (production rule memory), access the knowledge base, extract relevant production rules, and modify the knowledge base efficiently. Thus, in our example, the Design Search Algorithm is explicitly concerned with *computer-assisted* design. Table 20.1 shows the process states

generated in the course of searching for a solution to the automobile design problem. At each step of the search the consequent parts of the production rules (listed in Section 20.1.2) are matched against the current state of the design (the list of attributes in OPEN) – the latter comprising both the current structural attributes and the (remaining) current specifications. Since many production rules may match the current state, the preferred rule is the one that matches the first leftmost (remaining) of the sub-specifications in the list OPEN.

Table 12.1 Design Search Algorithm Applied to the Automobile Design Problem

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	r_{18}, r_{10}, r_{12}	\emptyset	2, 3, 4	4
2	r_{19}, r_{10}, r_{12}	r_{18}	6	6
3	$r_{22}, r_{27}, r_{21}, m_{10}, r_{10}, r_{12}$	r_{19}, r_{18}	11	11
4	$m_4, m_5, m_8, r_{27}, r_{21}, m_{10}, r_{10}, r_{12}$	r_{22}, r_{19}, r_{18}	12	12
5	$m_4, m_5, m_8, m_{18}, r_{21}, m_{10}, r_{10}, r_{12}$	$r_{27}, r_{22}, r_{19}, r_{18}$	13	13
6	$m_4, m_5, m_8, m_{18}, m_{19}, m_{35}, m_{10}, r_{10}, r_{12}$	$r_{21}, r_{27}, r_{22}, r_{19}, r_{18}$	15	15
7	$m_4, m_5, m_8, m_{18}, m_{19}, m_{35}, m_{10}, r_1, r_{30}, m_{23}, m_{27}, r_{12}$	$r_{10}, r_{21}, r_{27}, r_{22}, r_{19}, r_{18}$	16	16
8	$m_4, m_5, m_8, m_{18}, m_{19}, m_{35}, m_{10}, m_{29}, m_7, r_{30}, m_{23}, m_{27}, r_{12}$	$r_1, r_{10}, r_{21}, r_{27}, r_{22}, r_{19}, r_{18}$	17	17
9	$m_4, m_5, m_8, m_{18}, m_{19}, m_{35}, m_{10}, m_{29}, m_7, r_{11}, r_2, m_{34}, m_{30}, m_{23}, m_{27}, r_{12}$	$r_{30}, r_1, r_{10}, r_{21}, r_{27}, r_{22}, r_{19}, r_{18}$	18	18

10	$m_4, m_5, m_8, m_{18},$ $m_{19}, m_{35}, m_{10},$ $m_{29}, m_7, m_{26},$ $m_3, r_2, m_{34},$ $m_{30}, m_{23}, m_{27},$ r_{12}	$r_{11}, r_{30}, r_1, r_{10},$ $r_{21}, r_{27}, r_{22}, r_{19},$ r_{18}	30	30
11	$m_4, m_5, m_8, m_{18},$ $m_{19}, m_{35}, m_{10},$ $m_{29}, m_7, m_{26},$ $m_3, m_{32}, m_{28},$ $m_{34}, m_{30}, m_{23},$ m_{27}, r_{12}	$r_2, r_{11}, r_{30}, r_1,$ $r_{10}, r_{21}, r_{27}, r_{22},$ r_{19}, r_{18}	21	21
12	$m_4, m_5, m_8, m_{18},$ $m_{19}, m_{35}, m_{10},$ $m_{29}, m_7, m_{26},$ $m_3, m_{32}, m_{28},$ $m_{34}, m_{30}, m_{23},$ m_{27}, r_5, m_{13}	$r_{12}, r_2, r_{11}, r_{30},$ $r_1, r_{10}, r_{21}, r_{27},$ r_{22}, r_{19}, r_{18}	23	23
13	$m_4, m_5, m_8, m_{18},$ $m_{19}, m_{35}, m_{10},$ $m_{29}, m_7, m_{26},$ $m_3, m_{32}, m_{28},$ $m_{34}, m_{30}, m_{23},$ $m_{27}, m_{16}, m_{21},$ m_{13} (consistent solution)	$r_5, r_{12}, r_2, r_{11},$ $r_{30}, r_1, r_{10}, r_{21},$ $r_{27}, r_{22}, r_{19}, r_{18}$	STOP	

20.2 FORKLIFT DESIGN EXAMPLE

A forklift is a small, human-operated, gasoline-powered or a battery-driven truck that moves materials between operations, or is used to load finished goods from a storage and retrieval system for delivery to customers' plants, where the goods become their raw materials. In order to enable operations managers to deliver parts as they are needed, thus reducing stockpiles of expensive inventories throughout the plant, forklift trucks have to be designed to satisfy several functional requirements such as passing narrow pickup rows, lifting carriages to high and low shelves, and lifting and carrying heavy and fragile carriages.

20.2.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES***The Design Description Properties***

The following structural attributes specify the configuration of actual forklifts:

- | | |
|---|--|
| (m_1) battery-powered | (m_{17}) long front-to-rear distance |
| (m_2) diesel powered | (m_{18}) low driving engine power capacity |
| (m_3) drum brakes | (m_{19}) low extended mast |
| (m_4) extra strong roof | (m_{20}) low height |
| (m_5) foot-operated brakes | (m_{21}) low lifting engine power capacity |
| (m_6) hand-operated brakes | (m_{22}) mast tilt has medium steep |
| (m_7) heavy weight | (m_{23}) mast tilt is steep |
| (m_8) high driving engine power capacity | (m_{24}) medium distance between forks |
| (m_9) high extended mast | (m_{25}) medium fork length |
| (m_{10}) high ground clearance | (m_{26}) medium front-to-rear distance |
| (m_{11}) high lifting engine power capacity | (m_{27}) medium retracted mast |
| (m_{12}) high retracted mast | (m_{28}) medium width |
| (m_{13}) hydraulic disk brakes | (m_{29}) narrow width |
| (m_{14}) light weight | (m_{30}) platform |
| (m_{15}) long distance between forks | (m_{31}) pushing position |
| (m_{16}) long fork length | (m_{32}) radial tire |

- | | |
|---|--------------------------------|
| (m_{33}) regular roof | (m_{37}) standing position |
| (m_{34}) short fork length | (m_{38}) strong roof |
| (m_{35}) sitting position | (m_{39}) tubeless tire |
| (m_{36}) small front-to-rear distance | (m_{40}) wide width |

The Specification Properties

The following functional attributes describe the requirements and constraints:

- | | |
|---|--|
| (r_1) ABILITY TO CLIMB HIGH
STEEP ENTRANCE ROAD | MODERATELY SAFE |
| (r_2) ABILITY TO CLIMB
MEDIUM STEEP ENTRANCE
ROAD | (r_{11}) HIGH ACCELERATION |
| (r_3) ABILITY TO LIFT
CARRIAGE TO HIGH
SHELVES | (r_{12}) HIGH CARRIAGE BALANCE |
| (r_4) ABILITY TO LIFT
CARRIAGE TO LOWER
SHELVES | (r_{13}) HIGH CARRIAGE CARRYING
ABILITY |
| (r_5) ABILITY TO LIFT
INDIVIDUAL BOXES | (r_{14}) HIGH CARRIAGE
LIFTING ABILITY |
| (r_6) ABILITY TO LIFT
PLATFORM WITH MATERIAL | (r_{15}) HIGH CARRIAGE
LIFTING STABILITY |
| (r_7) CONTINUAL WORK | (r_{16}) HIGH DRIVING SPEED |
| (r_8) FORKLIFT OPERATOR IS
HIGHLY SAFE | (r_{17}) HIGH LIFTING AND
LOWERING SPEED |
| (r_9) FORKLIFT OPERATOR IS
LOW SAFE | (r_{18}) HIGH OPERATING
UTILIZATION |
| (r_{10}) FORKLIFT OPERATOR IS | (r_{19}) HIGH REACH-ABILITY OF
THE FORK |
| | (r_{20}) HIGH REACH-ABILITY TO
DEEP SHELF |
| | (r_{21}) LARGE SPINNING RADIUS |

- | | |
|---|--|
| (r ₂₂) LIFTING AND CARRYING
FRAGILE CARRIAGE | (r ₃₈) MEDIUM CARRIAGE
BALANCE |
| (r ₂₃) LIFTING AND CARRYING
HEAVY CARRIAGE | (r ₃₉) MEDIUM DRIVING SPEED |
| (r ₂₄) LIFTING AND CARRYING
LIGHT CARRIAGE | (r ₄₀) MEDIUM REACH-ABILITY
OF THE FORK |
| (r ₂₅) LIFTING AND CARRYING
ROBUST CARRIAGE | (r ₄₁) MEDIUM SPINNING RADIUS |
| (r ₂₆) LONG DRIVING DISTANCE | (r ₄₂) MEDIUM REACH-ABILITY
OF THE FORK |
| (r ₂₇) LOW ACCELERATION | (r ₄₃) MEDIUM REACH-ABILITY
TO DEEP SHELF |
| (r ₂₈) LOW CARRIAGE BALANCE | (r ₄₄) NON-CONTINUAL WORK |
| (r ₂₉) LOW CARRIAGE
CARRYING ABILITY | (r ₄₅) PASSING IRREGULAR
SURFACE |
| (r ₃₀) LOW CARRIAGE
LIFTING ABILITY | (r ₄₆) PASSING LOW PLACES |
| (r ₃₁) LOW CARRIAGE
LIFTING STABILITY | (r ₄₇) PASSING MEDIUM
PICKUP ROWS |
| (r ₃₂) LOW DRIVING SPEED | (r ₄₈) PASSING NARROW
PICKUP ROWS |
| (r ₃₃) LOW LIFTING AND
LOWERING SPEED | (r ₄₉) PASSING OBSTACLES |
| (r ₃₄) LOW OPERATING
UTILIZATION | (r ₅₀) PASSING REGULAR
SURFACE |
| (r ₃₅) LOW REACH-ABILITY
OF THE FORK | (r ₅₁) PASSING WIDE
PICKUP ROWS |
| (r ₃₆) LOW REACH-ABILITY TO
DEEP SHELF | (r ₅₂) RELIABLE TIRE |
| (r ₃₇) MEDIUM ACCELERATION | (r ₅₃) SHORT DRIVING
DISTANCE |

(*r*₅₄) SMALL SPINNING
RADIUS

(*r*₅₆) WORKING IN LOW CROWDED
ENVIRONMENT

(*r*₅₅) WORKING IN HIGHLY
CROWDED ENVIRONMENT

(*r*₅₇) WORKING IN MODERATELY
CROWDED ENVIRONMENT

20.2.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the forklift design domain is expressed in terms of the following production rules:

(RULE 1) PASSING WIDE PICKUP ROWS

If the forklift truck has LARGE SPINNING RADIUS & wide width
Then the forklift truck is capable of PASSING WIDE PICKUP ROWS

Cause: (1) LARGE SPINNING RADIUS - the turning radius of the forklift truck has to be small enough in order to spin in the aisle; (2) wide width - the width of the forklift truck has to match the width between the pickup rows.

(RULE 2) PASSING MEDIUM PICKUP ROWS

If the forklift truck has MEDIUM SPINNING RADIUS & medium width
Then the forklift truck is capable of PASSING MEDIUM PICKUP ROWS

(RULE 3) PASSING NARROW PICKUP ROWS

If the forklift truck has SMALL SPINNING RADIUS & narrow width
Then the forklift truck is capable of PASSING NARROW PICKUP ROWS

(RULE 4) LARGE SPINNING RADIUS

If the forklift truck has long front-to-rear distance
Then the forklift truck has LARGE SPINNING RADIUS

Cause: the distance between the front and the rear wheels has to be long enough in order to have a large spinning radius.

(RULE 5) MEDIUM SPINNING RADIUS

If the forklift truck has medium front-to-rear distance
Then the forklift truck has MEDIUM SPINNING RADIUS

(RULE 6) SMALL SPINNING RADIUS

If the forklift truck has short front-to-rear distance
Then the forklift truck has SMALL SPINNING RADIUS

(RULE 7) ABILITY TO LIFT CARRIAGE TO HIGH SHELVES-1

If the forklift truck has HIGH CARRIAGE LIFTING STABILITY & high extended mast
Then the forklift truck has the ABILITY TO LIFT CARRIAGE TO HIGH SHELVES

Cause: the stability of the forklift while lifting and the masthead (when extended) have to be high, when the top pickup shelf is high.

(RULE 8) ABILITY TO LIFT CARRIAGE TO HIGH SHELVES-2

If the forklift truck has LOW CARRIAGE LIFTING STABILITY & low extended mast
Then the forklift truck has the ABILITY TO LIFT CARRIAGE TO HIGH SHELVES

Cause: the stability of the forklift while lifting and the masthead (when extended) have to be low, when the top pickup shelf is low.

(RULE 9) HIGH CARRIAGE LIFTING STABILITY

If the forklift truck has heavy weight
Then the forklift truck has HIGH CARRIAGE LIFTING STABILITY

(RULE 10) LOW CARRIAGE LIFTING STABILITY

If the forklift truck has light weight
Then the forklift truck has LOW CARRIAGE LIFTING STABILITY

(RULE 11) HIGH REACH-ABILITY TO DEEP SHELF

If the forklift truck has HIGH REACH-ABILITY OF THE FORK
Then the forklift truck has HIGH REACH-ABILITY TO DEEP SHELF

(RULE 12) MEDIUM REACH-ABILITY TO DEEP SHELF

If the forklift truck has MEDIUM REACH-ABILITY OF THE FORK
Then the forklift truck has MEDIUM REACH-ABILITY TO DEEP SHELF

(RULE 13) LOW REACH-ABILITY TO DEEP SHELF

If the forklift truck has LOW REACH-ABILITY OF THE FORK
Then the forklift truck has LOW REACH-ABILITY TO DEEP SHELF

(RULE 14) HIGH REACH-ABILITY OF THE FORK

If the fork length is long & the mast tilt is steep
Then the forklift truck has HIGH REACH-ABILITY OF THE FORK

(RULE 15) MEDIUM REACH-ABILITY OF THE FORK

If the fork length is medium & the mast tilt has medium steep
Then the forklift truck has MEDIUM REACH-ABILITY OF THE FORK

(RULE 16) LOW REACH-ABILITY OF THE FORK

If the fork length is short & the mast tilt has medium steep
Then the forklift truck has LOW REACH-ABILITY OF THE FORK

(RULE 17) LONG DRIVING DISTANCE

If the forklift truck has HIGH DRIVING SPEED & HIGH ACCELERATION
Then the forklift truck is used for LONG DRIVING DISTANCE

(RULE 18) SHORT DRIVING DISTANCE

If the forklift truck has LOW DRIVING SPEED & LOW ACCELERATION
Then the forklift truck is used for SHORT DRIVING DISTANCE

(RULE 19) HIGH DRIVING SPEED

If the forklift truck has light weight & high driving engine power capacity & the forklift truck is diesel powered
Then the forklift truck has HIGH DRIVING SPEED

(RULE 20) LOW DRIVING SPEED

If the forklift truck has heavy weight & low driving engine power capacity & the forklift truck is battery-powered
Then the forklift truck has LOW DRIVING SPEED

(RULE 21) HIGH ACCELERATION

If the forklift truck has high driving engine power capacity & the forklift truck is diesel powered
Then the forklift truck has HIGH ACCELERATION

(RULE 22) LOW ACCELERATION

If the forklift truck has heavy weight & low driving engine power capacity & the forklift truck is battery-powered
Then the forklift truck has LOW ACCELERATION

(RULE 23) ABILITY TO CLIMB HIGH STEEP ENTRANCE ROAD

If the forklift truck has HIGH CARRIAGE BALANCE & HIGH ACCELERATION
Then the forklift truck has ABILITY TO CLIMB HIGH STEEP ENTRANCE ROAD

Cause: the forklift truck should have high carriage balance on the fork and high acceleration when the angle of the entrance road to the warehouse is highly steep.

(RULE 24) ABILITY TO CLIMB MEDIUM STEEP ENTRANCE ROAD

If the forklift truck has MEDIUM CARRIAGE BALANCE & MEDIUM ACCELERATION
Then the forklift truck has ABILITY TO CLIMB MEDIUM STEEP ENTRANCE ROAD

Cause: the forklift truck should have medium carriage balance over the fork and medium acceleration when the angle of the entrance road to the warehouse is medium steep.

(RULE 25) HIGH CARRIAGE BALANCE

If the forklift truck has long fork length & long distance between forks
Then the forklift truck has HIGH CARRIAGE BALANCE

Cause: the forklift truck has high carriage balance over the fork when half the fork length is much longer than the center of gravity of the carriage.

(RULE 26) MEDIUM CARRIAGE BALANCE

If the forklift truck has medium fork length & medium distance between forks

Then the forklift truck has MEDIUM CARRIAGE BALANCE

Cause: the forklift truck has medium carriage balance over the fork when half the fork length is relatively longer than the center of gravity of the carriage.

(RULE 27) ABILITY TO LIFT CARRIAGE TO LOWER SHELVES-1

If the forklift truck has high retracted mast

Then the forklift truck has the ABILITY TO LIFT CARRIAGE TO LOWER SHELVES

Cause: the mast of the forklift truck should be highly retractable when the lowest pick up shelves are located very low.

(RULE 28) ABILITY TO LIFT CARRIAGE TO LOWER SHELVES-2

If the forklift truck has medium retracted mast

Then the forklift truck has the ABILITY TO LIFT CARRIAGE TO LOWER SHELVES

Cause: the mast of the forklift truck should be moderately retractable when the lowest pick up shelves are located somewhat low.

(RULE 29) PASSING LOW PLACES

If the forklift truck has low height

Then the forklift truck is capable of PASSING LOW PLACES

(RULE 30) PASSING REGULAR SURFACE

If the forklift truck has RELIABLE TIRE & drum brakes & foot-operated brakes

Then the forklift truck is capable of PASSING REGULAR SURFACE in the warehouse or outside it

(RULE 31) PASSING IRREGULAR SURFACE

If the forklift truck has RELIABLE TIRE & hydraulic disk brakes & hand-operated brakes

Then the forklift truck is capable of PASSING IRREGULAR SURFACE in the warehouse or outside it

(RULE 32) RELIABLE TIRE

If the forklift truck has tubeless tire & radial tire

Then the forklift truck has RELIABLE TIRE

(RULE 33) PASSING OBSTACLES

If the forklift truck has RELIABLE TIRE & high driving engine power capacity & light weight & high ground clearance & the forklift truck is diesel powered

Then the forklift truck is capable of PASSING OBSTACLES in the warehouse or outside it

Cause: the forklift truck should have a high ground clearance in order to avoid colliding in objects (such as waste) on the surface of the floor.

(RULE 34) WORKING IN HIGHLY CROWDED ENVIRONMENT

If the FORKLIFT OPERATOR IS HIGHLY SAFE & the forklift truck has LOW DRIVING SPEED & LOW ACCELERATION

Then the forklift truck is capable of WORKING IN HIGHLY CROWDED ENVIRONMENT

Cause: when there exist other trucks in the warehouse, the forklift operator should be safe, and the forklift truck should have low driving speed and low acceleration.

(RULE 35) FORKLIFT OPERATOR IS HIGHLY SAFE

If the forklift truck has extra strong roof & the forklift operator has a sitting position

Then the FORKLIFT OPERATOR IS HIGHLY SAFE

(RULE 36) WORKING IN MODERATELY CROWDED ENVIRONMENT

If the FORKLIFT OPERATOR IS MODERATELY SAFE & the forklift truck has MEDIUM DRIVING SPEED & MEDIUM ACCELERATION

Then the forklift truck is capable of WORKING IN MODERATELY CROWDED ENVIRONMENT

(RULE 37) FORKLIFT OPERATOR IS MODERATELY SAFE

If the forklift truck has strong roof & the forklift operator has a standing position

Then the FORKLIFT OPERATOR IS MODERATELY SAFE

(RULE 38) WORKING IN LOW CROWDED ENVIRONMENT

If the FORKLIFT OPERATOR IS LOW SAFE & the forklift truck has HIGH DRIVING SPEED & HIGH ACCELERATION
Then the forklift truck is capable of WORKING IN LOW CROWDED ENVIRONMENT

(RULE 39) FORKLIFT OPERATOR IS LOW SAFE

If the forklift truck has regular roof & the forklift operator has a pushing position
Then the FORKLIFT OPERATOR IS LOW SAFE

(RULE 40) LIFTING AND CARRYING HEAVY CARRIAGE

If the forklift truck has HIGH CARRIAGE LIFTING ABILITY & HIGH CARRIAGE CARRYING ABILITY
Then the forklift truck is capable of LIFTING AND CARRYING HEAVY CARRIAGE

(RULE 41) HIGH CARRIAGE LIFTING ABILITY

If the forklift truck has high lifting engine power capacity
Then the forklift truck has HIGH CARRIAGE LIFTING ABILITY

(RULE 42) HIGH CARRIAGE CARRYING ABILITY

If the forklift truck has HIGH CARRIAGE BALANCE & high driving engine power capacity & hydraulic disk brakes & the forklift truck is diesel powered
Then the forklift truck has HIGH CARRIAGE CARRYING ABILITY

(RULE 43) LIFTING AND CARRYING LIGHT CARRIAGE

If the forklift truck has LOW CARRIAGE LIFTING ABILITY & LOW CARRIAGE CARRYING ABILITY
Then the forklift truck is capable of LIFTING AND CARRYING LIGHT CARRIAGE

(RULE 44) LOW CARRIAGE LIFTING ABILITY

If the forklift truck has low lifting engine power capacity
Then the forklift truck has LOW CARRIAGE LIFTING ABILITY

(RULE 45) LOW CARRIAGE CARRYING ABILITY

If the forklift truck has LOW CARRIAGE BALANCE & low driving engine power capacity & hydraulic disk brakes & the forklift truck is battery-powered

Then the forklift truck has LOW CARRIAGE CARRYING ABILITY

(RULE 46) ABILITY TO LIFT PLATFORM WITH MATERIAL

If the forklift truck has platform & sitting position

Then the forklift truck has ABILITY TO LIFT PLATFORM WITH MATERIAL

(RULE 47) ABILITY TO LIFT INDIVIDUAL BOXES-1

If the forklift truck has a standing position

Then the forklift truck has ABILITY TO LIFT INDIVIDUAL BOXES

(RULE 48) ABILITY TO LIFT INDIVIDUAL BOXES-2

If the forklift truck has a pushing position

Then the forklift truck has ABILITY TO LIFT INDIVIDUAL BOXES

(RULE 49) LIFTING AND CARRYING FRAGILE CARRIAGE

If the forklift truck has LOW LIFTING AND LOWERING SPEED & LOW DRIVING SPEED

Then the forklift truck is capable of LIFTING AND CARRYING FRAGILE CARRIAGE

(RULE 50) LOW LIFTING AND LOWERING SPEED

If the forklift truck has low lifting engine power capacity

Then the forklift truck has LOW LIFTING AND LOWERING SPEED

(RULE 51) LIFTING AND CARRYING ROBUST CARRIAGE

If the forklift truck has HIGH LIFTING AND LOWERING SPEED & HIGH DRIVING SPEED

Then the forklift truck is capable of LIFTING AND CARRYING ROBUST CARRIAGE

(RULE 52) HIGH LIFTING AND LOWERING SPEED

If the forklift truck has high lifting engine power capacity

Then the forklift truck has HIGH LIFTING AND LOWERING SPEED

(RULE 53) HIGH OPERATING UTILIZATION

If the forklift truck is capable of CONTINUAL WORK

Then the forklift truck has HIGH OPERATING UTILIZATION

(RULE 54) LOW OPERATING UTILIZATION

If the forklift truck is used for NON-CONTINUAL WORK

Then the forklift truck has LOW OPERATING UTILIZATION

(RULE 55) CONTINUAL WORK

If the forklift truck has high driving engine power capacity &
the forklift truck is diesel powered

Then the forklift truck is used for CONTINUAL WORK

(RULE 56) NON-CONTINUAL WORK

If the forklift truck has low driving engine power capacity &
the forklift truck is battery-powered

Then the forklift truck is used for NON-CONTINUAL WORK

20.2.3 FORKLIFT TRUCK SYNTHESIS USING THE DESIGN SEARCH ALGORITHM (SEE CHAPTER 10.3)

Assume that the designer is faced with the problem of designing a forklift truck that is able to achieve the following specifications:

1. Lifting and carrying heavy carriages (r_{23});
2. Ability to lift carriages to high shelves (given the top pickup shelf is high) (r_3);
3. High operating utilization (r_{18});
4. Passing wide pickup rows (r_{51});
5. Ability to climb high steep entrance road (r_1)

Table 20.2 shows the process states generated in the course of searching for a solution to the forklift truck design problem.

Table 20.2 Design Search Algorithm Applied to the Forklift Truck Design Problem

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	$r_{23}, r_3, r_{18}, r_{51}, r_1$	\emptyset	40	40
2	$r_{14}, r_{13}, r_3, r_{18}, r_{51}, r_1$	r_{23}	41	41
3	$m_{11}, r_{13}, r_3, r_{18}, r_{51}, r_1$	r_{14}, r_{23}	42	42
4	$m_{11}, r_{12}, m_8, m_{13}, m_2, r_3, r_{18}, r_{51}, r_1$	r_{13}, r_{14}, r_{23}	25	25
5	$m_{11}, m_{16}, m_{15}, m_8, m_{13}, m_2, r_3, r_{18}, r_{51}, r_1$	$r_{12}, r_{13}, r_{14}, r_{23}$	7, 8	7
6	$m_{11}, m_{16}, m_{15}, m_8, m_{13}, m_2, r_{15}, m_9, r_{18}, r_{51}, r_1$	$r_3, r_{12}, r_{13}, r_{14}, r_{23}$	9	9
7	$m_{11}, m_{16}, m_{15}, m_8, m_{13}, m_2, m_7, m_9, r_{18}, r_{51}, r_1$	$r_{15}, r_3, r_{12}, r_{13}, r_{14}, r_{23}$	53	53
8	$m_{11}, m_{16}, m_{15}, m_8, m_{13}, m_2, m_7, m_9, r_7, r_{51}, r_1$	$r_{18}, r_{15}, r_3, r_{12}, r_{13}, r_{14}, r_{23}$	55	55
9	$m_{11}, m_{16}, m_{15}, m_8, m_{13}, m_2, m_7, m_9, r_{51}, r_1$	$r_7, r_{18}, r_{15}, r_3, r_{12}, r_{13}, r_{14}, r_{23}$	1	1
10	$m_{11}, m_{16}, m_{15}, m_8, m_{13}$	$r_{51}, r_7, r_{18}, r_{15}, r_3, r_{12}, r_{13}, r_{14}$	4	4

	$m_2, m_7, m_9,$ r_{21}, m_{40}, r_1	r_{23}		
11	$m_{11}, m_{16},$ $m_{15}, m_8, m_{13},$ $m_2, m_7, m_9,$ m_{17}, m_{40}, r_1	$r_{21}, r_{51}, r_7, r_{18},$ $r_{15}, r_3, r_{12}, r_{13},$ r_{14}, r_{23}	23	23
12	$m_{11}, m_{16},$ $m_{15}, m_8, m_{13},$ $m_2, m_7, m_9,$ m_{17}, m_{40}, r_{11}	$r_1, r_{21}, r_{51}, r_7,$ $r_{18}, r_{15}, r_3, r_{12},$ r_{13}, r_{14}, r_{23}	21	21
13	$m_{11}, m_{16},$ $m_{15}, m_8, m_{13},$ $m_2, m_7, m_9,$ m_{17}, m_{40} (consistent solution)	$r_{11}, r_1, r_{21}, r_{51},$ $r_7, r_{18}, r_{15}, r_3,$ $r_{12}, r_{13}, r_{14}, r_{23}$	STOP	

20.3 COMPUTER CLASSROOM DESIGN EXAMPLE

Due to the rapid changes in computer and other technologies, it is more important than ever for operation managers to keep abreast of the state of the art. Computer training programs are means of updating managers (medium to high level executives) on the rapid changes in computer technology. The high demand for this kind of training raises the issue of effective computer classroom design. The typical computer classroom is set up with rows of tables with computers, such that each computer can be used by an individual student while facing the teacher’s instruction area. The design of a computer classroom has to take into consideration functional requirements such as the level of ergonomics, equipment, safety, and protection against theft.

20.3.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES

The Design Description Properties

The following structural attributes specify the configuration of computer classrooms:

(m_1) 0.005 hp

(m_3) 1 fire extinguishers(3 kg each)

(m_2) 0.015 hp

- | | |
|---|---|
| (<i>m</i> ₄) 1 megabytes screen memory | (<i>m</i> ₂₃) 32 megabytes RAM size |
| (<i>m</i> ₅) 1.1 gigabytes disk size | (<i>m</i> ₂₄) 350 X 220 cm sized teacher's instruction area |
| (<i>m</i> ₆) 1.7 gigabytes disk size | (<i>m</i> ₂₅) 400 LUX |
| (<i>m</i> ₇) 100 btu per square meter | (<i>m</i> ₂₆) 450 x 250 cm sized teacher's instruction area |
| (<i>m</i> ₈) 100 x 60 cm. desk size | (<i>m</i> ₂₇) 8 megabytes RAM size |
| (<i>m</i> ₉) 101 keyboard keys | (<i>m</i> ₂₈) 80 X 60 cm. desk size |
| (<i>m</i> ₁₀) 104 keyboard keys | (<i>m</i> ₂₉) 850 megabytes disk size |
| (<i>m</i> ₁₁) 120 x 60 cm. desk size | (<i>m</i> ₃₀) acoustics walls with wood covered and graded ceiling |
| (<i>m</i> ₁₂) 13 m. visual distance | (<i>m</i> ₃₁) alarm |
| (<i>m</i> ₁₃) 14" screen memory | (<i>m</i> ₃₂) attached locks |
| (<i>m</i> ₁₄) 15" screen memory | (<i>m</i> ₃₃) bars |
| (<i>m</i> ₁₅) 16 megabytes RAM size | (<i>m</i> ₃₄) common slide viewer |
| (<i>m</i> ₁₆) 164 btu per square meter | (<i>m</i> ₃₅) common wooden desk |
| (<i>m</i> ₁₇) 17" screen memory | (<i>m</i> ₃₆) computerized slide viewer |
| (<i>m</i> ₁₈) 2 fire extinguishers(3 kg each) | (<i>m</i> ₃₇) custom made desks |
| (<i>m</i> ₁₉) 2 megabytes screen memory | (<i>m</i> ₃₈) executive chair |
| (<i>m</i> ₂₀) 20 - 50 cm. platform's height | (<i>m</i> ₃₉) IBM compatible mouse (3 keys) |
| (<i>m</i> ₂₁) 2300 lumens fluorescent tubes | (<i>m</i> ₄₀) illumination at 90 degree |
| (<i>m</i> ₂₂) 280 X 200 cm. sized teacher's instruction area | |

- | | |
|--|---|
| (m_{41}) integrated wooden office system | (m_{56}) secondary passage = 100 cm. |
| (m_{42}) locked cabinets | (m_{57}) secondary passage = 70 cm. |
| (m_{43}) Logitech mouse | (m_{58}) secondary passage = 85 cm. |
| (m_{44}) low radiation screen | (m_{59}) secretary chair |
| (m_{45}) non-interlaced screen | (m_{60}) standard chalk board |
| (m_{46}) Novel network | (m_{61}) standard walls painted in acoustic paint |
| (m_{47}) original Microsoft mouse | (m_{62}) standard wood office desk |
| (m_{47}) Pentium 100 MHz | (m_{63}) steel construction |
| (m_{49}) Pentium 120 MHz | (m_{64}) tin desk |
| (m_{46}) Pentium 133 MHz | (m_{65}) tin table |
| (m_{51}) Pentium 200 MHz | (m_{66}) waiting chair |
| (m_{45}) primary passage = 120 cm. | (m_{67}) white board with erasable markers |
| (m_{49}) primary passage = 150 cm. | (m_{68}) Windows NT network |
| (m_{54}) primary passage = 180 cm. | |
| (m_{55}) regular wood door | |

The Specification Properties

The following functional attributes describe the requirements and constraints:

- | | |
|---|---|
| (r_1) GOOD ILLUMINATION | (r_3) GOOD ILLUMINATION AT STUDENTS WORK AREA |
| (r_2) GOOD ILLUMINATION AT TEACHER'S INSTRUCTION AREA | (r_4) GOOD VISUAL CONTACT |
| | (r_5) HIGH LEVEL ACOUSTICS |

- (r_6) HIGH LEVEL AIR
CONDITIONING
- (r_7) HIGH LEVEL CENTRAL
PROCESSING UNIT (CPU)
- (r_8) HIGH LEVEL CHAIRS AT
TEACHER'S INSTRUCTION
AREA
- (r_9) HIGH LEVEL CHAIRS AT
STUDENTS WORK AREA
- (r_{10}) HIGH LEVEL COMPUTER
CLASSROOM
- (r_{11}) HIGH LEVEL COMPUTER
NETWORK
- (r_{12}) HIGH LEVEL COMPUTER
SYSTEM
- (r_{13}) HIGH LEVEL DESKS AT
TEACHING INSTRUCTION
AREA
- (r_{14}) HIGH LEVEL DESKS AT
STUDENTS WORK AREA
- (r_{15}) HIGH LEVEL EQUIPMENT
- (r_{16}) HIGH LEVEL ERGONOMICS
- (r_{17}) HIGH LEVEL FIRE SAFETY
- (r_{18}) HIGH LEVEL FURNITURE
- (r_{19}) HIGH LEVEL FURNITURE AT
STUDENTS WORK AREA
- (r_{20}) HIGH LEVEL FURNITURE AT
TEACHER'S INSTRUCTION
AREA
- (r_{21}) HIGH LEVEL HARD DISK
DRIVE
- (r_{22}) HIGH LEVEL KEYBOARD TYPE
- (r_{23}) HIGH LEVEL MOUSE TYPE
- (r_{24}) HIGH LEVEL PERSONAL
COMPUTERS
- (r_{25}) HIGH LEVEL PROTECTION
AGAINST THEFT
- (r_{26}) HIGH LEVEL RANDOM ACCESS
MEMORY
- (r_{27}) HIGH LEVEL SAFE DOOR
- (r_{28}) HIGH LEVEL SAFE
EQUIPMENT
- (r_{29}) HIGH LEVEL SAFE WINDOW
- (r_{30}) HIGH LEVEL SAFETY
- (r_{31}) HIGH LEVEL SCREEN
CONTROLLER
- (r_{32}) HIGH LEVEL SCREEN
RADIATION SAFETY
- (r_{33}) HIGH LEVEL SCREEN SIZE
- (r_{34}) HIGH LEVEL SLIDE VIEWER
- (r_{35}) HIGH LEVEL SPACIOUS
CLASSROOM
- (r_{36}) HIGH LEVEL SPACIOUS
PASSAGES
- (r_{37}) HIGH LEVEL SPACIOUS
TEACHER'S INSTRUCTION
AREA
- (r_{38}) HIGH LEVEL SPACIOUS
STUDENTS WORK AREA

- | | |
|--|--|
| (r ₃₉) HIGH LEVEL STUDYING AND TEACHING EQUIPMENT | (r ₅₅) LOW LEVEL PERSONAL COMPUTERS |
| (r ₄₀) HIGH LEVEL TEACHER'S WRITING BOARD | (r ₅₆) LOW LEVEL RANDOM ACCESS MEMORY |
| (r ₄₁) LOW LEVEL CENTRAL PROCESSING UNIT (CPU) | (r ₅₇) LOW LEVEL SAFETY |
| (r ₄₂) LOW LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA | (r ₅₈) LOW LEVEL SCREEN RADIATION SAFETY |
| (r ₄₃) LOW LEVEL CHAIRS AT STUDENTS WORK AREA | (r ₅₉) LOW LEVEL SCREEN SIZE |
| (r ₄₄) LOW LEVEL COMPUTER CLASSROOM | (r ₆₀) LOW LEVEL SPACIOUS CLASSROOM |
| (r ₄₅) LOW LEVEL COMPUTER SYSTEM | (r ₆₁) LOW LEVEL SPACIOUS PASSAGES |
| (r ₄₆) LOW LEVEL DESKS AT TEACHING INSTRUCTION AREA | (r ₆₂) LOW LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA |
| (r ₄₇) LOW LEVEL DESKS AT STUDENTS WORK AREA | (r ₆₃) LOW LEVEL SPACIOUS STUDENTS WORK AREA |
| (r ₄₈) LOW LEVEL EQUIPMENT | (r ₆₄) LOW LEVEL STUDYING AND TEACHING EQUIPMENT |
| (r ₄₉) LOW LEVEL ERGONOMICS | (r ₆₅) MEDIUM LEVEL CHAIRS AT STUDENTS WORK AREA |
| (r ₅₀) LOW LEVEL FURNITURE | (r ₆₆) MEDIUM LEVEL DESKS AT STUDENTS WORK AREA |
| (r ₅₁) LOW LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA | (r ₆₇) MEDIUM LEVEL ACOUSTICS |
| (r ₅₂) LOW LEVEL FURNITURE AT STUDENTS WORK AREA | (r ₆₈) MEDIUM LEVEL AIR CONDITIONING |
| (r ₅₃) LOW LEVEL HARD DISK DRIVE | (r ₆₉) MEDIUM LEVEL CENTRAL PROCESSING UNIT (CPU) |
| (r ₅₄) LOW LEVEL MOUSE TYPE | (r ₇₀) MEDIUM LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA |

- | | |
|---|--|
| (r ₇₁) MEDIUM LEVEL CHAIRS AT STUDENTS WORK AREA | (r ₈₇) MEDIUM LEVEL PERSONAL COMPUTERS |
| (r ₇₂) MEDIUM LEVEL COMPUTER CLASSROOM | (r ₈₈) MEDIUM LEVEL PROTECTION AGAINST THEFT |
| (r ₇₃) MEDIUM LEVEL COMPUTER NETWORK | (r ₈₉) MEDIUM LEVEL RANDOM ACCESS MEMORY |
| (r ₇₄) MEDIUM LEVEL COMPUTER SYSTEM | (r ₉₀) MEDIUM LEVEL SAFE DOOR |
| (r ₇₅) MEDIUM LEVEL DESKS AT TEACHING INSTRUCTION AREA | (r ₉₁) MEDIUM LEVEL SAFE EQUIPMENT |
| (r ₇₆) MEDIUM LEVEL DESKS AT STUDENTS WORK AREA | (r ₉₂) MEDIUM LEVEL SAFE WINDOW |
| (r ₇₇) MEDIUM LEVEL EQUIPMENT | (r ₉₃) MEDIUM LEVEL SAFETY |
| (r ₇₈) MEDIUM LEVEL ERGONOMICS | (r ₉₄) MEDIUM LEVEL SCREEN CONTROLLER |
| (r ₇₉) MEDIUM LEVEL FIRE SAFETY | (r ₉₅) MEDIUM LEVEL SCREEN RADIATION SAFETY |
| (r ₈₀) MEDIUM LEVEL FURNITURE | (r ₉₆) MEDIUM LEVEL SCREEN SIZE |
| (r ₈₁) MEDIUM LEVEL FURNITURE AT STUDENTS WORK AREA | (r ₉₇) MEDIUM LEVEL SLIDE VIEWER |
| (r ₈₂) MEDIUM LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA | (r ₉₈) MEDIUM LEVEL SPACIOUS CLASSROOM |
| (r ₈₃) MEDIUM LEVEL FURNITURE AT STUDENTS WORK AREA | (r ₉₉) MEDIUM LEVEL SPACIOUS PASSAGES |
| (r ₈₄) MEDIUM LEVEL HARD DISK DRIVE | (r ₁₀₀) MEDIUM LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA |
| (r ₈₅) MEDIUM LEVEL KEYBOARD TYPE | (r ₁₀₁) MEDIUM LEVEL SPACIOUS STUDENTS WORK AREA |
| (r ₈₆) MEDIUM LEVEL MOUSE TYPE | (r ₁₀₂) MEDIUM LEVEL STUDYING AND TEACHING EQUIPMENT |
| | (r ₁₀₃) MEDIUM LEVEL TEACHER'S WRITING BOARD |

20.3.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the computer classroom design domain is expressed in terms of the following production rules:

(RULE 1) HIGH LEVEL COMPUTER CLASSROOM

If the classroom has HIGH LEVEL ERGONOMICS & HIGH LEVEL EQUIPMENT & HIGH LEVEL SAFETY & HIGH LEVEL PROTECTION AGAINST THEFT

Then the classroom is a HIGH LEVEL COMPUTER CLASSROOM

Cause: high protection against theft is used when there are high theft hazards.

(RULE 2) MEDIUM LEVEL COMPUTER CLASSROOM

If the classroom has MEDIUM LEVEL ERGONOMICS & MEDIUM LEVEL EQUIPMENT & MEDIUM LEVEL SAFETY & MEDIUM LEVEL PROTECTION AGAINST THEFT

Then the classroom is a MEDIUM LEVEL COMPUTER CLASSROOM

(RULE 3) LOW LEVEL COMPUTER CLASSROOM

If the classroom has LOW LEVEL ERGONOMICS & LOW LEVEL EQUIPMENT & LOW LEVEL SAFETY & MEDIUM LEVEL PROTECTION AGAINST THEFT

Then the classroom is a LOW LEVEL COMPUTER CLASSROOM

Cause: medium protection against theft is used when there are low theft hazards.

(RULE 4) HIGH LEVEL ERGONOMICS

If the classroom has GOOD ILLUMINATION & HIGH LEVEL FURNITURE & GOOD VISUAL CONTACT & HIGH LEVEL ACOUSTICS & is HIGH LEVEL SPACIOUS

Then the computer classroom has HIGH LEVEL ERGONOMICS

Cause: an ergonomic classroom provides good studying and teaching environment.

(RULE 5) MEDIUM LEVEL ERGONOMICS

If the classroom has GOOD ILLUMINATION & MEDIUM LEVEL FURNITURE & GOOD VISUAL CONTACT & MEDIUM LEVEL ACOUSTICS & the classroom is MEDIUM LEVEL SPACIOUS

Then the computer classroom has MEDIUM LEVEL ERGONOMICS

(RULE 6) LOW LEVEL ERGONOMICS

If the classroom has GOOD ILLUMINATION & LOW LEVEL FURNITURE & GOOD VISUAL CONTACT & MEDIUM LEVEL ACOUSTICS & is LOW LEVEL SPACIOUS

Then the computer classroom has LOW LEVEL ERGONOMICS

(RULE 7) HIGH LEVEL SPACIOUS CLASSROOM

If HIGH LEVEL SPACIOUS STUDENTS WORK AREA & HIGH LEVEL SPACIOUS PASSAGES & HIGH LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Then HIGH LEVEL SPACIOUS CLASSROOM

(RULE 8) MEDIUM LEVEL SPACIOUS CLASSROOM

If MEDIUM LEVEL SPACIOUS STUDENTS WORK AREA & MEDIUM LEVEL SPACIOUS PASSAGES & MEDIUM LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Then MEDIUM LEVEL SPACIOUS classroom

(RULE 9) LOW LEVEL SPACIOUS CLASSROOM

If LOW LEVEL SPACIOUS STUDENTS WORK AREA & LOW LEVEL SPACIOUS PASSAGES & LOW LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Then LOW LEVEL SPACIOUS classroom

(RULE 10) HIGH LEVEL SPACIOUS STUDENTS WORK AREA

If 120 x 60 cm. desk size

Then HIGH LEVEL SPACIOUS STUDENTS WORK AREA

Cause: a student's desk size is 120 x 60 cm. for maximal sized students work area

(RULE 11) MEDIUM LEVEL SPACIOUS STUDENTS WORK AREA

If 100 x 60 cm. desk size

Then MEDIUM LEVEL SPACIOUS STUDENTS WORK AREA

Cause: a student's desk size is 100 x 60 cm. for medium sized students work area.

(RULE 12) LOW LEVEL SPACIOUS STUDENTS WORK AREA

If 80 X 60 cm. desk size

Then LOW LEVEL SPACIOUS STUDENTS WORK AREA

Cause: a student's desk size of 80 X 60 cm. is used for minimal sized students work area.

(RULE 13) HIGH LEVEL SPACIOUS PASSAGES

If primary passage = 180 cm. & secondary passage = 100 cm.

Then HIGH LEVEL SPACIOUS PASSAGES

Cause: for courses oriented for high level personnel (medium to high level executives) use maximum spacing (width) between blocks of tables (primary passage) and maximum space between rows of tables (secondary passage).

(RULE 14) MEDIUM LEVEL SPACIOUS PASSAGES

If primary passage = 150 cm. & secondary passage = 85 cm.

Then MEDIUM LEVEL SPACIOUS PASSAGES

Cause: for courses oriented for medium level personnel (low level executives), use medium spacing between blocks of tables (primary passage) and space between rows of tables (secondary passage).

(RULE 15) LOW LEVEL SPACIOUS PASSAGES

If primary passage = 120 cm. & secondary passage = 70 cm.

Then LOW LEVEL SPACIOUS PASSAGES

Cause: for courses oriented for low level personnel (standard courses), use minimal spacing between blocks of tables (primary passage) and space between rows of tables (secondary passage).

(RULE 16) HIGH LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

If 450 x 250 cm sized teacher's instruction area

Then HIGH LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Cause: for courses oriented for high level personnel (medium to high level executives), use maximal sized teacher's instruction area.

(RULE 17) MEDIUM LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

If 350 X 220 cm sized teacher's instruction area

Then MEDIUM LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Cause: for courses oriented for medium level personnel (low level executives), use medium sized teacher's instruction area.

(RULE 18) LOW LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

If 280 X 200 cm. sized teacher's instruction area
Then LOW LEVEL SPACIOUS TEACHER'S INSTRUCTION AREA

Cause: for courses oriented for low level personnel (standard courses), use minimal sized teacher's instruction area.

(RULE 19) HIGH LEVEL ACOUSTICS

If acoustics walls with wood covered and graded
ceiling
Then HIGH LEVEL ACOUSTICS

Cause: for courses oriented for medium to high level personnel (courses oriented for medium to high level executives), use high level acoustics, which provides high noise absorption and good hearing quality.

(RULE 20) MEDIUM LEVEL ACOUSTICS

If standard walls painted in acoustic paint
Then MEDIUM LEVEL ACOUSTICS

Cause: for courses oriented for low to medium level personnel (standard courses up to one's oriented for low level executives), use standard level acoustics, which provides normal noise absorption and standard hearing quality.

(RULE 21) GOOD ILLUMINATION

If GOOD ILLUMINATION AT TEACHER'S INSTRUCTION AREA &
GOOD ILLUMINATION AT STUDENTS WORK AREA
Then GOOD ILLUMINATION

Cause: (1) ILLUMINATION AT TEACHER'S INSTRUCTION AREA - high lighting intensity and angle at teacher's instruction area provide good visual ability students to teaching means; (2) ILLUMINATION AT STUDENTS WORK AREA - high lighting intensity and angle at student's desk provide good visual ability at student's desk for writing and reading activities.

(RULE 22) GOOD ILLUMINATION AT TEACHER'S INSTRUCTION AREA

If 2300 lumens fluorescent tubes & at 90 degree (ceiling to floor)

Then GOOD ILLUMINATION AT TEACHER'S INSTRUCTION AREA

(RULE 23) GOOD ILLUMINATION AT STUDENTS WORK AREA

If 400 LUX & 90 degree (ceiling to floor)

Then GOOD ILLUMINATION AT STUDENTS WORK AREA

(RULE 24) HIGH LEVEL FURNITURE

If HIGH LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA
& HIGH LEVEL FURNITURE AT STUDENTS WORK AREA

Then HIGH LEVEL FURNITURE

Cause: comfortable furniture for teacher and students create good concentration of teaching and studying.

(RULE 25) MEDIUM LEVEL FURNITURE

If MEDIUM LEVEL FURNITURE AT TEACHER'S INSTRUCTION
AREA & MEDIUM LEVEL FURNITURE AT STUDENTS WORK AREA

Then MEDIUM LEVEL FURNITURE

(RULE 26) LOW LEVEL FURNITURE

If LOW LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA &
LOW LEVEL FURNITURE AT STUDENTS WORK AREA

Then LOW LEVEL FURNITURE

(RULE 27) HIGH LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA

If HIGH LEVEL DESKS AT TEACHING INSTRUCTION AREA &
HIGH LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Then HIGH LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA

(RULE 28) MEDIUM LEVEL FURNITURE AT TEACHER'S INSTRUCTION
AREA

If MEDIUM LEVEL DESKS AT TEACHING INSTRUCTION AREA &
MEDIUM LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Then MEDIUM LEVEL FURNITURE AT TEACHER'S INSTRUCTION
AREA

(RULE 29) LOW LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA

If LOW LEVEL DESKS AT TEACHING INSTRUCTION AREA & LOW
LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Then LOW LEVEL FURNITURE AT TEACHER'S INSTRUCTION AREA

(RULE 30) HIGH LEVEL DESKS AT TEACHING INSTRUCTION AREA

If integrated wooden office system (selection from catalog serial numbers 950 - 954)

Then HIGH LEVEL DESKS AT TEACHING INSTRUCTION AREA

Cause: integrated wooden office system is a high quality desk.

(RULE 31) MEDIUM LEVEL DESKS AT TEACHING INSTRUCTION AREA

If standard wood office desk (selection from catalog serial numbers 300 - 311, 840 - 841)

Then MEDIUM LEVEL DESKS AT TEACHING INSTRUCTION AREA

Cause: standard wood office desk is a medium quality desk.

(RULE 32) LOW LEVEL DESKS AT TEACHING INSTRUCTION AREA

If tin table (selecting from catalog serial numbers 600 - 604)

Then LOW LEVEL DESKS AT TEACHING INSTRUCTION AREA

Cause: tin desk is a common quality desk.

(RULE 33) HIGH LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

If executive chair (selection from catalog serial numbers 202 - 215)

Then HIGH LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Cause: executive chair is a high quality chair.

(RULE 34) MEDIUM LEVEL CHAIRS AT TEACHER INSTRUCTION AREA

If secretary chair (selection from catalog serial numbers 101 - 118)

Then MEDIUM LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Cause: secretary chair is a medium quality chair.

(RULE 35) LOW LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

If waiting chair (selection from catalog serial numbers 119 -133 or 134 -137)

Then LOW LEVEL CHAIRS AT TEACHER'S INSTRUCTION AREA

Cause: waiting chair is a common quality chair.

(RULE 36) HIGH LEVEL FURNITURE AT STUDENTS WORK AREA

If HIGH LEVEL DESKS AT STUDENTS WORK AREA & HIGH LEVEL CHAIRS AT STUDENTS WORK AREA
 Then HIGH LEVEL FURNITURE AT STUDENTS WORK AREA

(RULE 37) MEDIUM LEVEL FURNITURE AT STUDENTS WORK AREA

If MEDIUM LEVEL DESKS AT STUDENTS WORK AREA & MEDIUM LEVEL CHAIRS AT STUDENTS WORK AREA
 Then MEDIUM LEVEL FURNITURE AT STUDENTS WORK AREA

(RULE 38) HIGH LEVEL DESKS AT STUDENTS WORK AREA

If common wooden desk (selection from catalog serial numbers 300 - 311 or 840 - 841)
 Then HIGH LEVEL DESKS AT STUDENTS WORK AREA

Cause: wooden desk is a high quality desk.

(RULE 39) MEDIUM LEVEL DESKS AT STUDENTS WORK AREA

If tin desk (selection from catalog serial numbers 600 - 604)
 Then MEDIUM LEVEL DESKS AT STUDENTS WORK AREA

Cause: tin desk is a medium quality desk.

(RULE 40) LOW LEVEL DESKS AT STUDENTS WORK AREA

If custom made desks (hand made by special carpenter order)
 Then LOW LEVEL DESKS AT STUDENTS WORK AREA

Cause: custom made desk (long desk with several computers) is a common quality desk.

(RULE 41) HIGH LEVEL CHAIRS AT STUDENTS WORK AREA

If executive chair (selection from catalog serial numbers 202 - 215)
 Then HIGH LEVEL CHAIRS AT STUDENTS WORK AREA

Cause: executive chair is a high quality chair.

(RULE 42) MEDIUM LEVEL CHAIRS AT STUDENTS WORK AREA

If secretary chair (selection from catalog serial numbers 101 - 118)

Then MEDIUM LEVEL CHAIRS AT STUDENTS WORK AREA

Cause: secretary chair is a medium quality chair.

(RULE 43) LOW LEVEL CHAIRS AT STUDENTS WORK AREA

If waiting chair (selection from catalog serial numbers 119 -133 or 134 -137)

Then LOW LEVEL CHAIRS AT STUDENTS WORK AREA

Cause: waiting chair is a common quality chair.

(RULE 44) GOOD VISUAL CONTACT

If 20 - 50 cm. platform's height & 13 m. visual distance

Then GOOD VISUAL CONTACT

Cause: the recommended teacher's platform height and visual distance (distance between teacher's board and the most remote desk) provide appropriate visual contact between the teacher and the students.

(RULE 45) HIGH LEVEL EQUIPMENT

If HIGH LEVEL STUDYING AND TEACHING EQUIPMENT & HIGH LEVEL AIR CONDITIONING

Then HIGH LEVEL EQUIPMENT

(RULE 46) MEDIUM LEVEL EQUIPMENT

If MEDIUM LEVEL STUDYING AND TEACHING EQUIPMENT & MEDIUM LEVEL AIR CONDITIONING

Then MEDIUM LEVEL EQUIPMENT

(RULE 47) LOW LEVEL EQUIPMENT

If LOW LEVEL STUDYING AND TEACHING EQUIPMENT & MEDIUM LEVEL AIR CONDITIONING

Then LOW LEVEL EQUIPMENT

(RULE 48) HIGH LEVEL STUDYING AND TEACHING EQUIPMENT

If HIGH LEVEL TEACHER'S WRITING BOARD & HIGH LEVEL SLIDE VIEWER & HIGH LEVEL COMPUTER SYSTEM

Then HIGH LEVEL STUDYING AND TEACHING EQUIPMENT

(RULE 49) MEDIUM LEVEL STUDYING AND TEACHING EQUIPMENT

If MEDIUM LEVEL TEACHER'S WRITING BOARD & MEDIUM LEVEL
SLIDE VIEWER & MEDIUM LEVEL COMPUTER SYSTEM
Then MEDIUM LEVEL STUDYING AND TEACHING EQUIPMENT

(RULE 50) LOW LEVEL STUDYING AND TEACHING EQUIPMENT

If MEDIUM LEVEL TEACHER'S WRITING BOARD & MEDIUM LEVEL
SLIDE VIEWER & LOW LEVEL COMPUTER SYSTEM
Then LOW LEVEL STUDYING AND TEACHING EQUIPMENT

(RULE 51) HIGH LEVEL TEACHER'S WRITING BOARD

If white board with erasable markers
Then HIGH LEVEL TEACHER'S WRITING BOARD

Cause: white board with erasable markers in a high quality board that provides dust reduction , and is easy to use and colorful.

(RULE 52) MEDIUM LEVEL TEACHER'S WRITING BOARD

If standard chalk board
Then MEDIUM LEVEL TEACHER'S WRITING BOARD

Cause: standard chalk board is a common board that requires low maintenance cost.

(RULE 53) HIGH LEVEL SLIDE VIEWER

If common slide viewer & computerized slide viewer
Then HIGH LEVEL SLIDE VIEWER

Cause: common slide viewer & computerized slide viewer enable to view both simple and computerized slides, thus providing high quality teaching utility enhancement.

(RULE 54) MEDIUM LEVEL SLIDE VIEWER

If common slide viewer
Then MEDIUM LEVEL SLIDE VIEWER

Cause: common slide viewer provides common quality teaching utility enhancement.

(RULE 55) HIGH LEVEL COMPUTER SYSTEM-1

If HIGH LEVEL PERSONAL COMPUTERS & HIGH LEVEL COMPUTER NETWORK
Then HIGH LEVEL COMPUTER SYSTEM

Cause: a suitable computer system is the integration of PC's and a network that meets the needs of teaching & software requirements.

(RULE 56) MEDIUM LEVEL COMPUTER SYSTEM-1

If MEDIUM LEVEL PERSONAL COMPUTERS & MEDIUM LEVEL COMPUTER NETWORK
Then MEDIUM LEVEL COMPUTER SYSTEM

(RULE 57) LOW LEVEL COMPUTER SYSTEM-1

If LOW LEVEL PERSONAL COMPUTERS & MEDIUM LEVEL COMPUTER NETWORK
Then LOW LEVEL COMPUTER SYSTEM

(RULE 58) HIGH LEVEL COMPUTER SYSTEM-2

If HIGH LEVEL PERSONAL COMPUTERS
Then HIGH LEVEL COMPUTER SYSTEM

Cause: the integration of PC's with a network is expensive. Thus, despite the need to connect PC's on a network that will enable network's study, a suitable choice might be to renounce the computer network installation.

(RULE 59) MEDIUM LEVEL COMPUTER SYSTEM-2

If MEDIUM LEVEL PERSONAL COMPUTERS
Then MEDIUM LEVEL COMPUTER SYSTEM

Cause: see "(RULE 58) HIGH LEVEL COMPUTER SYSTEM-2"

(RULE 60) LOW LEVEL COMPUTER SYSTEM-2

If LOW LEVEL PERSONAL COMPUTERS
Then LOW LEVEL COMPUTER SYSTEM

Cause: see "(RULE 58) HIGH LEVEL COMPUTER SYSTEM-2"

(RULE 61) HIGH LEVEL PERSONAL COMPUTERS

If HIGH LEVEL RANDOM ACCESS MEMORY & HIGH LEVEL HARD DISK DRIVE & HIGH LEVEL CPU & HIGH LEVEL SCREEN CONTROLLER & HIGH LEVEL SCREEN SIZE & HIGH LEVEL KEYBOARD TYPE & HIGH LEVEL MOUSE TYPE
 Then HIGH LEVEL PERSONAL COMPUTERS

(RULE 62) MEDIUM LEVEL PERSONAL COMPUTERS

If MEDIUM LEVEL RANDOM ACCESS MEMORY & MEDIUM LEVEL HARD DISK DRIVE & MEDIUM LEVEL CPU & MEDIUM LEVEL SCREEN CONTROLLER & MEDIUM LEVEL SCREEN SIZE & MEDIUM LEVEL KEYBOARD TYPE & MEDIUM LEVEL MOUSE TYPE
 Then MEDIUM LEVEL PERSONAL COMPUTERS

(RULE 63) LOW LEVEL PERSONAL COMPUTERS

If LOW LEVEL RANDOM ACCESS MEMORY & LOW LEVEL HARD DISK DRIVE & LOW LEVEL CPU & MEDIUM LEVEL SCREEN CONTROLLER & LOW LEVEL SCREEN SIZE & MEDIUM LEVEL KEYBOARD TYPE & LOW LEVEL MOUSE TYPE
 Then LOW LEVEL PERSONAL COMPUTERS

(RULE 64) HIGH LEVEL RANDOM ACCESS MEMORY

If 32 megabytes RAM size
 Then HIGH LEVEL RANDOM ACCESS MEMORY

Cause: 32 megabytes RAM provides high standard performance (the bigger the RAM, the higher the performance).

(RULE 65) MEDIUM LEVEL RANDOM ACCESS MEMORY

If 16 megabytes RAM size
 Then MEDIUM LEVEL RANDOM ACCESS MEMORY

Cause: 16 megabytes RAM provides medium standard performance (the bigger the RAM, the higher the performance).

(RULE 66) LOW LEVEL RANDOM ACCESS MEMORY

If 8 megabytes RAM size
 Then LOW LEVEL RANDOM ACCESS MEMORY

Cause: 8 megabytes RAM provides low standard performance (the bigger the RAM, the higher the performance).

(RULE 67) HIGH LEVEL HARD DISK DRIVE

If 1.7 gigabytes disk size
Then HIGH LEVEL HARD DISK DRIVE

Cause: 1.7 gigabytes hard disk drive provides high standard performance (the bigger the disk, the higher the performance).

(RULE 68) MEDIUM LEVEL HARD DISK DRIVE

If 1.1 gigabytes disk size
Then MEDIUM LEVEL HARD DISK DRIVE

Cause: 1.1 gigabytes hard disk drive provides medium standard performance (the bigger the disk, the higher the performance).

(RULE 69) LOW LEVEL HARD DISK DRIVE

If 850 megabytes disk size
Then LOW LEVEL HARD DISK DRIVE

Cause: 850 megabytes hard disk drive provides low standard performance (the bigger the disk, the higher the performance).

(RULE 70) HIGH LEVEL CENTRAL PROCESSING UNIT

If Pentium 200 MHz
Then HIGH LEVEL CENTRAL PROCESSING UNIT(CPU)

Cause: Pentium 200 MHz CPU provides very high standard performance of CPU (the faster the CPU, the higher the performance).

(RULE 71) MEDIUM LEVEL CENTRAL PROCESSING UNIT-1

If Pentium 133 MHz
Then MEDIUM LEVEL CENTRAL PROCESSING UNIT(CPU)

Cause: Pentium 133 MHz CPU provides high standard performance (the faster the CPU, the higher the performance).

(RULE 72) MEDIUM LEVEL CENTRAL PROCESSING UNIT-2

If Pentium 120 MHz
Then MEDIUM LEVEL CENTRAL PROCESSING UNIT(CPU)

Cause: Pentium 120 MHz CPU provides medium standard performance (the faster the CPU, the higher the performance).

(RULE 73) LOW LEVEL CENTRAL PROCESSING UNIT

If Pentium 100 MHz
Then LOW LEVEL CENTRAL PROCESSING UNIT(CPU)

Cause: Pentium 100 MHz CPU provides low standard performance (the faster the CPU, the higher the performance).

(RULE 74) HIGH LEVEL SCREEN CONTROLLER

If 2 megabytes screen memory
Then HIGH LEVEL SCREEN CONTROLLER

Cause: 2 megabytes of screen controller memory provides high standard performance (the more megabytes of screen controller memory; the higher its performance).

(RULE 75) MEDIUM LEVEL SCREEN CONTROLLER

If 1 megabytes screen memory
Then MEDIUM LEVEL SCREEN CONTROLLER

Cause: 1 megabyte of screen controller memory provides medium standard performance (the more megabytes of screen controller memory; the higher its performance).

(RULE 76) HIGH LEVEL SCREEN SIZE

If 17" screen memory
Then HIGH LEVEL SCREEN SIZE

Cause: 17" screen size provides high standard vision performance (the bigger the screen the better the vision).

(RULE 77) MEDIUM LEVEL SCREEN SIZE

If 15" screen memory
Then MEDIUM LEVEL SCREEN SIZE

Cause: 15" screen size provides medium standard vision performance (the bigger the screen the better the vision).

(RULE 78) LOW LEVEL SCREEN SIZE

If 14" screen memory
Then LOW LEVEL SCREEN SIZE

Cause: 14" screen size provides low standard vision performance (the bigger the screen the better the vision).

(RULE 79) HIGH LEVEL KEYBOARD TYPE

If 104 keyboard keys
Then HIGH LEVEL KEYBOARD TYPE

Cause: new standard keyboard requirements (for windows 95).

(RULE 80) MEDIUM LEVEL KEYBOARD TYPE

If 101 keyboard keys
Then MEDIUM LEVEL KEYBOARD TYPE

Cause: standard keyboard requirements.

(RULE 81) HIGH LEVEL MOUSE TYPE

If original Microsoft mouse
Then HIGH LEVEL MOUSE TYPE

Cause: original (ergonomic) Microsoft mouse provides high standard performance.

(RULE 82) MEDIUM LEVEL MOUSE TYPE

If Logitech mouse
Then MEDIUM LEVEL MOUSE TYPE

Cause: LOGITECH mouse provides medium standard performance.

(RULE 83) LOW LEVEL MOUSE TYPE

If IBM compatible(3 keys)
Then LOW LEVEL MOUSE TYPE

Cause: IBM compatible (3 keys) mouse provides low standard performance.

(RULE 84) HIGH LEVEL COMPUTER NETWORK

If Windows NT
Then HIGH LEVEL COMPUTER NETWORK

Cause: Windows NT is a high graphic interface computer network that requires low computer resources.

(RULE 85) MEDIUM LEVEL COMPUTER NETWORK

If Novel network
Then MEDIUM LEVEL COMPUTER NETWORK

Cause: Novel network is a low graphic interface computer network that requires low computer resources.

(RULE 86) HIGH LEVEL AIR CONDITIONING .

If 0.015 hp & 164 btu per square meter
Then HIGH LEVEL AIR CONDITIONING

Cause: large size classroom requires high standard air conditioning equipment, which creates a comfortable environment for the teacher and students by climate control.

(RULE 87) MEDIUM LEVEL AIR CONDITIONING

If 0.005 hp & 100 btu per square meter
Then MEDIUM LEVEL AIR CONDITIONING

Cause: medium size classroom requires medium standard air conditioning equipment.

(RULE 88) HIGH LEVEL SAFETY

If HIGH LEVEL SCREEN RADIATION SAFETY & HIGH LEVEL
FIRE SAFETY
Then HIGH LEVEL SAFETY

(RULE 89) MEDIUM LEVEL SAFETY

If MEDIUM LEVEL SCREEN RADIATION SAFETY & MEDIUM LEVEL
FIRE SAFETY
Then MEDIUM LEVEL SAFETY

(RULE 90) LOW LEVEL SAFETY

If LOW LEVEL SCREEN RADIATION SAFETY & MEDIUM LEVEL
 FIRE SAFETY
Then LOW LEVEL SAFETY

(RULE 91) HIGH LEVEL SCREEN RADIATION SAFETY

If non-interlaced screen & low radiation screen
Then HIGH LEVEL SCREEN RADIATION SAFETY

Cause: non-interlaced screen & low radiation screen are used when there are high standards regarding radiation.

(RULE 92) MEDIUM LEVEL SCREEN RADIATION SAFETY

If low radiation screen
Then MEDIUM LEVEL SCREEN RADIATION SAFETY

Cause: low radiation screen is used when there are medium standards regarding radiation.

(RULE 93) LOW LEVEL SCREEN RADIATION SAFETY

If non-interlaced screen
Then LOW LEVEL SCREEN RADIATION SAFETY

Cause: non-interlaced screen is used when there are low radiation standards.

(RULE 94) HIGH LEVEL FIRE SAFETY

If 2 fire extinguishers(3 kg each)
Then HIGH LEVEL FIRE SAFETY

Cause: 2 fire extinguishers (3 kg each) are used when there are high standards regarding fire safety.

(RULE 95) MEDIUM LEVEL FIRE SAFETY

If 1 fire extinguishers(3 kg each)
Then MEDIUM LEVEL FIRE SAFETY

Cause: 1 fire extinguisher (3 kg each) is used when there are normal standards regarding fire safety.

(RULE 96) HIGH LEVEL PROTECTION AGAINST THEFT

If HIGH LEVEL SAFE WINDOW & HIGH LEVEL SAFE DOOR & HIGH
LEVEL SAFE EQUIPMENT
Then HIGH LEVEL PROTECTION AGAINST THEFT

(RULE 97) MEDIUM LEVEL PROTECTION AGAINST THEFT

If MEDIUM LEVEL SAFE WINDOW & MEDIUM LEVEL SAFE DOOR &
MEDIUM LEVEL SAFE EQUIPMENT
Then MEDIUM LEVEL PROTECTION AGAINST THEFT

(RULE 98) HIGH LEVEL SAFE WINDOW

If bars
Then HIGH LEVEL SAFE WINDOW

(RULE 99) HIGH LEVEL SAFE DOOR

If steel construction
Then HIGH LEVEL SAFE DOOR

(RULE 100) HIGH LEVEL SAFE EQUIPMENT

If attached locks
Then HIGH LEVEL SAFE EQUIPMENT

(RULE 101) MEDIUM LEVEL SAFE WINDOW

If alarm
Then MEDIUM LEVEL SAFE WINDOW

(RULE 102) MEDIUM LEVEL SAFE DOOR

If regular wood door
Then MEDIUM LEVEL SAFE DOOR

(RULE 103) MEDIUM LEVEL SAFE EQUIPMENT

If locked cabinets
Then MEDIUM LEVEL SAFE EQUIPMENT

20.3.3 COMPUTER CLASSROOM SYNTHESIS USING THE DESIGN SEARCH ALGORITHM (SEE CHAPTER 10.3)

Assume that the designer is faced with the problem of designing a high level computer classroom (r_{10}). Table 20.3 shows part of the design process generated in the course of searching for a solution to the computer classroom design problem.

Table 20.3 The Design Search Algorithm Applied to the Computer Classroom Design Problem

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	r_{10}	\emptyset	1	1
2	$r_{16}, r_{15}, r_{30}, r_{25}$	r_{10}	4	4
3	$r_1, r_{18}, r_4, r_5, r_{35}, r_{15}, r_{30}, r_{25}$	r_{16}, r_{10}	21	21
4	$r_2, r_3, r_{18}, r_4, r_5, r_{35}, r_{15}, r_{30}, r_{25}$	r_1, r_{16}, r_{10}	22	22
5	$m_{21}, m_{40}, r_3, r_{18}, r_4, r_5, r_{35}, r_{15}, r_{30}, r_{25}$	r_2, r_1, r_{16}, r_{10}	23	23
6	$m_{21}, m_{40}, m_{25}, r_{18}, r_4, r_5, r_{35}, r_{15}, r_{30}, r_{25}$	$r_3, r_2, r_1, r_{16}, r_{10}$	24	24
7	$m_{21}, m_{40}, m_{25}, r_{20}, r_{19}, r_4, r_5, r_{35}, r_{15}, r_{30}, r_{25}$	$r_{18}, r_3, r_2, r_1, r_{16}, r_{10}$	27	27
8	$m_{21}, m_{40}, m_{25}, r_{13}, r_8, r_{19}, r_4,$	$r_{20}, r_{18}, r_3, r_2, r_1, r_{16}, r_{10}$	30	30

	$r_5, r_{35}, r_{15},$ r_{30}, r_{25}			
9	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $r_8, r_{19}, r_4,$ $r_5, r_{35}, r_{15},$ r_{30}, r_{25}	$r_{13}, r_{20}, r_{18}, r_3,$ r_2, r_1, r_{16}, r_{10}	33	33
10	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, r_{19}, r_4,$ $r_5, r_{35}, r_{15},$ r_{30}, r_{25}	$r_8, r_{13}, r_{20}, r_{18},$ $r_3, r_2, r_1, r_{16}, r_{10}$	36	36
11	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, r_{14}, r_9,$ $r_4, r_5, r_{35},$ r_{15}, r_{30}, r_{25}	$r_{19}, r_8, r_{13}, r_{20},$ $r_{18}, r_3, r_2, r_1, r_{16},$ r_{10}	38	38
12	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $r_9, r_4, r_5,$ $r_{35}, r_{15}, r_{30},$ r_{25}	$r_{14}, r_{19}, r_8, r_{13},$ $r_{20}, r_{18}, r_3, r_2, r_1,$ r_{16}, r_{10}	41	41
13	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $r_4, r_5, r_{35},$ r_{15}, r_{30}, r_{25}	$r_9, r_{14}, r_{19}, r_8,$ $r_{13}, r_{20}, r_{18}, r_3,$ r_2, r_1, r_{16}, r_{10}	44	44
14	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $m_{20}, r_5, r_{35},$ r_{15}, r_{30}, r_{25}	$r_4, r_9, r_{14}, r_{19}, r_8,$ $r_{13}, r_{20}, r_{18}, r_3,$ r_2, r_1, r_{16}, r_{10}	19	19
15	$m_{21}, m_{40},$ $m_{25}, m_{41},$	$r_5, r_4, r_9, r_{14}, r_{19},$ $r_8, r_{13}, r_{20}, r_{18},$	7	7

	$m_{38}, m_{35},$ $m_{20}, m_{30},$ $r_{35}, r_{15}, r_{30},$ r_{25}	$r_3, r_2, r_1, r_{16}, r_{10}$		
16	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $m_{20}, m_{30},$ $r_{38}, r_{36}, r_{37},$ r_{15}, r_{30}, r_{25}	$r_{35}, r_5, r_4, r_9, r_{14},$ $r_{19}, r_8, r_{13}, r_{20},$ $r_{18}, r_3, r_2, r_1, r_{16},$ r_{10}	10	10
17	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $m_{20}, m_{30},$ $m_{11}, r_{36},$ $r_{37}, r_{15}, r_{30},$ r_{25}	$r_{38}, r_{35}, r_5, r_4, r_9,$ $r_{14}, r_{19}, r_8, r_{13},$ $r_{20}, r_{18}, r_3, r_2, r_1,$ r_{16}, r_{10}	13	13
18	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $m_{20}, m_{30},$ $m_{11}, m_{54},$ $r_{37}, r_{15}, r_{30},$ r_{25}	$r_{36}, r_{38}, r_{35}, r_5,$ $r_4, r_9, r_{14}, r_{19}, r_8,$ $r_{13}, r_{20}, r_{18}, r_3,$ r_2, r_1, r_{16}, r_{10}	16	16
19	$m_{21}, m_{40},$ $m_{25}, m_{41},$ $m_{38}, m_{35},$ $m_{20}, m_{30},$ $m_{11}, m_{54},$ $m_{26}, r_{15},$ r_{30}, r_{25} (partial consistent solution)	$r_{37}, r_{36}, r_{38}, r_{35},$ $r_5, r_4, r_9, r_{14}, r_{19},$ $r_8, r_{13}, r_{20}, r_{18},$ $r_3, r_2, r_1, r_{16}, r_{10}$	45	45

20.4 TIRE DESIGN EXAMPLE

Tires are rubber-and-fabric devices that, when attached to the wheels of a vehicle, provide the contact between the vehicle and the surface over which it travels. Tires may be either solid or pneumatic (air-filled) in structure, with the latter by far the most prevalent today. Each modern automotive tire supports 50 times its own weight. Compressed air within the tire carries 90 percent of the load, with the tire's complex structure of rubber and fabric carrying the remaining 10 percent.

Construction

The most visible parts of a tire are the tread, which grips the road surface, and the supporting sidewalls, which run from tread to wheel rim. Tread patterns are especially important when the road is wet. The forward portion of a tire's contact patch wipes away water so that the rest of the patch grips a drier surface. Continuous channels from the center to the edge of the tread direct the water outward. Without a carefully designed tread, water would form a wedge and cause the tire to lift off the road. This so-called aquaplaning phenomenon is one reason that smooth tires (whether they are intentionally smooth racing slicks or regular tires that have been worn bald) are dangerous in wet conditions. Snow tires and off-highway tires have deeper treads or separate cleats that bite through snow, slush, or dirt to grip the firmer surface beneath.

A tire's sidewalls have two conflicting purposes: they flex up and down, helping cushion the vehicle from road irregularities, yet they must be relatively rigid horizontally in order to transfer loads of steering, braking, and acceleration. At their innermost edges, sidewalls meet the tire's beads (hoops of steel wire covered with hard rubber). Each bead reinforces the interface between the tire and wheel-rim and fixes the tire's inner diameter.

Reinforcing cords, which give the tire its strength, are arranged beneath the tire's surface. The three classes of modern tires can be distinguished by the direction of the cords: (1) a *bias-ply* tire (the earliest) has two or more plies of cord running across the tire at an angle, or bias, from the tire's centerline; the cords thus form a crisscross pattern from bead to bead; (2) in a *belted-bias* tire, reinforcing belts are placed between the plies; (3) a *radial* tire has reinforcing cords running hoop-fashion from bead to bead. Like a bias-belted tire, a radial tire has reinforcing belts under its tread, but radial belt cords are angled closer to the tire's centerline. The lack of bias sidewall reinforcement makes a radial's sidewalls more flexible. This gives the tread a better grip and longer life; it also gives the radial tire its characteristic underinflated look.

The tire construction begins with the sidewall. The different rubber compounds necessitated by the different requirements of each part of the tire are brought together to form sidewall-tread-sidewall strips. Beads are formed from wound and rubber-coated steel wires. Steel or synthetic-fiber fabric is rubber-coated and cut either at an angle or straight across, depending on tire type. All these materials are assembled for hand lay-up on a rotating, collapsible drum. The operator carefully aligns reinforcing

fabric over a rubber liner. An error of 1 degree in cord angle is noticeable; a 2 degree error can cause a tire to be scrapped. Next, the beads and tread-sidewall strip are added, and the result is an uncured, or green, tire. A heated press molds the tread pattern and vulcanizes the rubber.

Tire Dimensions

The basic tire dimensions are defined as follows:

- *Rim Width* - the linear distance between the flanges of the rim.
- *Overall Diameter* - the diameter of an inflated tire at the outermost surface of the tread.
- *Rim Diameter* - the diameter at the intersection of the planes of the rim bead seat and the rim flange.
- *Section Height* - Half the difference between the overall diameter and the rim diameter.
- *Section Width* - the linear distance between the outsides of the sidewalls of an inflated tire excluding elevations due to labeling (markings), decorations, or protective bands or ribs.
- *Aspect Ratio* - one hundred times the ratio of the section height to the section width of the tire on its rim.

Tire Designation

Tires come in many sizes, each described by a coded sequence. For example, the Tire Size Designation P 165/75 R 13 (approved by the International Standards Organization) is a passenger-car (P), radial (R), 165 mm. section width, 75 percent aspect ratio (profile), and a 13 in. rim diameter code. The number 75 describes the tire's cross-sectional profile: its height is 75 percent of its width; the cross-section of 60 percent aspect ratio would be somewhat more squat. Because of different handling characteristics, tires of differing size or type should not be mixed.

In addition to the Tire Size Designation, a tire may be identified by a Service Description consisting of a *load index* and a *speed symbol*. The load index is a numerical code associated with the maximum load a tire can carry for speed up to and including 210 Km/h under the basic inflation pressure of 2.5 bar. The inflation pressure means the pressure taken with the tire at ambient temperature and does not include any pressure build-up due to tire usage. The speed symbol indicates the speed limit for the tire. Speed limit means the maximum speed of which the car is capable. The common load indices, speed symbols and the corresponding load and speed capabilities of the vehicle are shown in Table 20.4.

Table 20.4 Load Capabilities and Speed Limits Corresponding to the Service Description

Speed Symbol - maximum Km/h	H - 210	Load Index - maximum Kg per tire	89 - 580
	U - 200		88 - 560
	T - 190		87 - 545
	S - 180		86 - 530
	R - 170		85 - 515
	Q - 160		84 - 500
	P - 150		83 - 487
	N - 140		82 - 475
	M - 130		81 - 462
			80 - 450
			79 - 437
			78 - 425
	77 - 412		
	76 - 400		
	75 - 387		

20.4.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES***The Design Description Properties***

The design structural attributes specify the configuration of a tire:

- | | |
|---|---|
| (m_1) 135 or 145 or 155
or 165 width symbol | (m_7) 82 or 80 or 70 aspect
ratio symbol |
| (m_2) 195 or 185 or 175
or 165 width symbol | (m_8) 85 or 84 or 83 or 82
or 81 or 80 load
capacity symbol |
| (m_3) 215 or 205 or 195
width symbol | (m_9) 89 or 88 or 87 or 86
or 85 load capacity
symbol |
| (m_4) 50 or 55 or 60
aspect ratio symbol | (m_{10}) H or U or T speed
symbol |
| (m_5) 60 or 65 or 70
aspect ratio symbol | (m_{11}) Q or P or N or M
speed symbol |
| (m_6) 80 or 79 or 78 or 77
or 76 or 75 load
capacity symbol | (m_{12}) T or S or R or Q
speed symbol |

The Specification Properties

The following functional attributes describe the requirements and constraints:

- | | |
|------------------------------------|---|
| (r_1) DRIVING COMFORT | (r_{20}) LOW WIDTH |
| (r_2) EXECUTIVE DRIVING | (r_{21}) LOW ASPECT RATIO |
| (r_3) FAMILY DRIVING | (r_{22}) LOW DRIVING SPEED |
| (r_4) GOOD TRACTION | (r_{23}) LOW FUEL CONSUMPTION |
| (r_5) HIGH ASPECT RATIO | (r_{24}) LOW LOAD CAPABILITY |
| (r_6) HIGH ASPECT RATIO | (r_{25}) LOW NOISE |
| (r_7) HIGH COMFORT | (r_{26}) LOW SPEED CAPABILITY |
| (r_8) HIGH DRIVING COMFORT | (r_{27}) LOW WIDTH |
| (r_9) HIGH DRIVING SPEED | (r_{28}) MATERIAL
TRANSPORTATION |
| (r_{10}) HIGH LOAD CAPABILITY | (r_{29}) MEDIUM WIDTH |
| (r_{11}) HIGH LOAD CAPACITY | (r_{30}) MEDIUM ASPECT RATIO |
| (r_{12}) HIGH SAFETY | (r_{31}) MEDIUM DRIVING SPEED |
| (r_{13}) HIGH SPEED CAPABILITY | (r_{32}) MEDIUM LOAD CAPABILITY |
| (r_{14}) HIGH TRACTION | (r_{33}) MEDIUM LOAD CAPACITY |
| (r_{15}) HIGH WIDTH | (r_{34}) MEDIUM NOISE |
| (r_{16}) LONG DISTANCE DRIVING | (r_{35}) MEDIUM SPEED
CAPABILITY |
| (r_{17}) LOW ASPECT RATIO | (r_{36}) MEDIUM TRACTION |
| (r_{18}) LOW FUEL CONSUMPTION | |
| (r_{19}) LOW NOISE | |

(r_{37}) MEDIUM WIDTH

(r_{40}) SPORT DRIVING

(r_{38}) REGULAR COMFORT

(r_{41}) TAXICAB

(r_{39}) REGULAR DRIVING
COMFORT

(r_{42}) URBAN DRIVING

20.4.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the tire design domain is expressed in terms of the following production rules:

(RULE 1) HIGH ASPECT RATIO

If the tire has 82 or 80 or 70 aspect ratio symbol
Then the tire has a HIGH ASPECT RATIO

(RULE 2) HIGH LOAD CAPABILITY

If the tire has 89 or 88 or 87 or 86 or 85 load capacity symbol
Then the tire has a HIGH LOAD CAPABILITY

(RULE 3) HIGH SPEED CAPABILITY

If the tire has H or U or T speed symbol
Then the tire has a HIGH SPEED CAPABILITY

(RULE 4) HIGH WIDTH

If the tire has 215 or 205 or 195 width symbol
Then the tire has a HIGH WIDTH

(RULE 5) LOW WIDTH

If the tire has 135 or 145 or 155 or 165 width symbol
Then the tire has a LOW WIDTH

(RULE 6) LOW ASPECT RATIO

If the tire has 50 or 55 or 60 aspect ratio symbol
Then the tire has a LOW ASPECT RATIO

(RULE 7) LOW LOAD CAPABILITY

If the tire has 80 or 79 or 78 or 77 or 76 or 75 load capacity symbol

Then the tire has a LOW LOAD CAPABILITY

(RULE 8) LOW SPEED CAPABILITY

If the tire has Q or P or N or M speed symbol

Then the tire has a LOW SPEED CAPABILITY

(RULE 9) MEDIUM LOAD CAPABILITY

If the tire has 85 or 84 or 83 or 82 or 81 or 80 load capacity symbol

Then the tire has a MEDIUM LOAD CAPABILITY

(RULE 10) MEDIUM ASPECT RATIO

If the tire has 60 or 65 or 70 aspect ratio symbol

Then the tire has a MEDIUM ASPECT RATIO

(RULE 11) MEDIUM SPEED CAPABILITY

If the tire has T or S or R or Q speed symbol

Then the tire has a MEDIUM SPEED CAPABILITY

(RULE 12) MEDIUM WIDTH

If the tire has 195 or 185 or 175 or 165 width symbol

Then the tire has a MEDIUM WIDTH

(RULE 13) MEDIUM TRACTION

If the tire has MEDIUM WIDTH & LOW ASPECT RATIO & MEDIUM SPEED CAPABILITY

Then the car has MEDIUM TRACTION

(RULE 14) HIGH LOAD CAPACITY

If the tire has HIGH LOAD CAPABILITY & HIGH WIDTH

Then the car has HIGH LOAD CAPACITY

(RULE 15) HIGH TRACTION

If the tire has HIGH WIDTH & LOW ASPECT RATIO & HIGH SPEED CAPABILITY
Then the car has HIGH TRACTION

(RULE 16) HIGH DRIVING SPEED

If the tire has HIGH WIDTH & LOW ASPECT RATIO & HIGH SPEED CAPABILITY
Then the car has HIGH DRIVING SPEED

(RULE 17) LOW NOISE

If the tire has LOW WIDTH & HIGH ASPECT RATIO
Then the car has LOW NOISE

(RULE 18) LOW FUEL CONSUMPTION

If the tire has LOW WIDTH & MEDIUM ASPECT RATIO
Then the car has LOW FUEL CONSUMPTION

(RULE 19) LOW DRIVING SPEED

If the tire has LOW WIDTH & MEDIUM ASPECT RATIO & LOW SPEED CAPABILITY
Then the car has LOW DRIVING SPEED

(RULE 20) MEDIUM DRIVING SPEED

If the tire has MEDIUM SPEED CAPABILITY & MEDIUM WIDTH & MEDIUM ASPECT RATIO
Then the car has MEDIUM DRIVING SPEED

(RULE 21) MEDIUM LOAD CAPACITY

If the tire has MEDIUM LOAD CAPABILITY & MEDIUM WIDTH
Then the car has MEDIUM LOAD CAPACITY

(RULE 22) MEDIUM NOISE

If the tire has MEDIUM WIDTH & MEDIUM ASPECT RATIO
Then the car has MEDIUM NOISE

(RULE 23) REGULAR DRIVING COMFORT

If the tire has MEDIUM WIDTH & MEDIUM ASPECT RATIO
Then the car has REGULAR DRIVING COMFORT

(RULE 24) HIGH DRIVING COMFORT

If the tire has HIGH ASPECT RATIO & HIGH WIDTH
Then the car has HIGH DRIVING COMFORT

(RULE 25) HIGH SAFETY

If the car has HIGH TRACTION
Then the car has HIGH SAFETY

(RULE 26) HIGH COMFORT

If the car has HIGH DRIVING COMFORT & LOW NOISE
Then the car has HIGH COMFORT

(RULE 27) LONG DISTANCE DRIVING

If the car has MEDIUM DRIVING SPEED & LOW NOISE & HIGH
DRIVING COMFORT
Then the car enables LONG DISTANCE DRIVING

(RULE 28) REGULAR COMFORT

If the car has MEDIUM NOISE & REGULAR DRIVING COMFORT
Then the car has REGULAR COMFORT

(RULE 29) SPORT DRIVING

If the tire has LOW ASPECT RATIO & HIGH WIDTH & the car has HIGH
DRIVING SPEED
Then the car is used for SPORT DRIVING

(RULE 30) URBAN DRIVING

If the car has LOW DRIVING SPEED & DRIVING COMFORT & LOW
FUEL CONSUMPTION
Then the car is used for URBAN DRIVING

(RULE 31) FAMILY DRIVING

If the car has HIGH COMFORT & GOOD TRACTION & MEDIUM DRIVING SPEED & MEDIUM LOAD CAPACITY
Then the car is used for FAMILY DRIVING

(RULE 32) EXECUTIVE DRIVING

If the car has HIGH COMFORT & HIGH DRIVING SPEED & MEDIUM LOAD CAPACITY
Then the car is used for EXECUTIVE DRIVING

(RULE 33) MATERIAL TRANSPORTATION

If the car has HIGH LOAD CAPACITY & LOW DRIVING SPEED & REGULAR COMFORT
Then the car is used for MATERIAL TRANSPORTATION

(RULE 34) TAXICAB

If the car has HIGH DRIVING COMFORT & LOW FUEL CONSUMPTION & LONG DISTANCE DRIVING
Then the car is used as TAXICAB

***20.4.3 TIRE SYNTHESIS USING THE DESIGN SEARCH ALGORITHM
(SEE CHAPTER 10.3)***

Assume that the designer is faced with the problem of designing a tire for a car that is used for material transportation (r_{28}). Table 20.5 shows the process states generated while searching for a solution to the tire design problem. As shown in Table 20.5 (process step 13), the functional requirement r_{28} cannot be satisfied. This situation is likely to occur when the number of structural attributes (in this case, the tire's structural properties) in an artifact (i.e., automobile) is much less than the number of functional requirements. In this case, the inconsistency may be resolved by the addition of new design attributes (e.g., engine structural properties) to make the number of structural attributes equal to the number of functional requirements.

Table 20.5 Design Search Algorithm Applied to the Forklift Truck Design Problem

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	r_{28}	\emptyset	33	33
2	r_{11}, r_{22}, r_{38}	r_{28}	14	14
3	$r_{10}, r_{15}, r_{22}, r_{38}$	r_{11}, r_{28}	2	2
4	$m_9, r_{15}, r_{22}, r_{38}$	r_{10}, r_{11}, r_{28}	4	4
5	m_9, m_3, r_{22}, r_{38}	$r_{15}, r_{10}, r_{11}, r_{28}$	19	19
6	$m_9, m_3, r_{27}, r_{30}, r_{26}, r_{38}$	$r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	5	5
7	$m_9, m_3, m_1, r_{30}, r_{26}, r_{38}$	$r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	10	10
8	$m_9, m_3, m_1, m_5, r_{26}, r_{38}$	$r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	8	8
9	$m_9, m_3, m_1, m_5, m_{11}, r_{38}$	$r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	28	28
10	$m_9, m_3, m_1, m_5, m_{11}, r_{34}, r_{31}$	$r_{38}, r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	22	22
11	$m_9, m_3, m_1, m_5, m_{11}, r_{29}, r_{31}$	$r_{34}, r_{38}, r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	12	12
12	$m_9, m_3, m_1, m_5, m_{11}, m_2, r_{31}$	$r_{29}, r_{34}, r_{38}, r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	23	23
13	$m_9, m_3, m_1, m_5, m_{11}, m_2$ ($m_1, m_2, \& m_3$ are inconsistent)	$r_{31}, r_{29}, r_{34}, r_{38}, r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	CONFLICT RESOLUTION (selecting only one structural attribute)	
14	m_9, m_3, m_5, m_{11} (consistent solution)	$r_{31}, r_{29}, r_{34}, r_{38}, r_{26}, r_{30}, r_{27}, r_{22}, r_{15}, r_{10}, r_{11}, r_{28}$	STOP	

20.5 FASTENER DESIGN EXAMPLE

The design of mechanical fasteners is a common design problem in the realm of mechanical engineering (see Chapter 3). The almost infinite variety of mechanical, over two million different kinds, has caused some manufacturers who have large assemblies to carefully re-evaluate the function of each fastener, with the view of better standardization. Fasteners may be divided into five main types: threaded, rivets, washers and retaining rings, pin fasteners, and quick-operating fasteners.

20.5.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES

The Design Description Properties

The design description properties specify the configuration of the actual fasteners:

- | | |
|---|---|
| (m_1) 5 sided head | (m_{15}) nut |
| (m_2) alloy | (m_{16}) Phillips driving recess |
| (m_3) aluminum | (m_{17}) Phillips head |
| (m_4) aluminum alloy | (m_{18}) Phillips slot |
| (m_5) break away head | (m_{19}) plug in socket |
| (m_6) chamfer point | (m_{20}) slot driving recess |
| (m_7) coarse thread | (m_{21}) spanner head |
| (m_8) copper alloy | (m_{22}) stainless steel |
| (m_9) cotter pin | (m_{23}) thread that was rolled
after heat treatment |
| (m_{10}) fine thread | (m_{24}) titanium |
| (m_{11}) hexagon driving recess | (m_{25}) twelve point head bolt |
| (m_{12}) hexagon head | (m_{26}) washer |
| (m_{13}) internal wrenching
hex socket | (m_{27}) wire |
| (m_{14}) non-metallic material | |

The Specification Properties

The following functional attributes describe the requirements and constraints:

- | | |
|---|---|
| (r_1) BOLT TYPE | (r_{15}) STANDARD DRIVING RECESS |
| (r_2) CORROSION RESISTANCE | (r_{16}) STRENGTH TO WEIGHT RATIO |
| (r_3) DIFFICULTY OF DISASSEMBLY | (r_{17}) USED FOR FATIGUE LOADING |
| (r_4) EASE OF DISASSEMBLY | (r_{18}) USED IN AIRCRAFT INDUSTRY |
| (r_5) ELECTRICAL AND THERMAL CONDUCTIVITY | (r_{19}) USED IN ELECTRICAL INDUSTRY |
| (r_6) HIGH SHEAR STRENGTH | (r_{20}) USED IN HIGH OPERATION TEMPERATURE |
| (r_7) HIGH TENSILE STRENGTH | (r_{21}) USED IN VIBRATING ATMOSPHERE |
| (r_8) LOW WEIGHT | (r_{22}) USES PNEUMATIC WRENCH |
| (r_9) PRECISION | (r_{23}) USES PNEUMATIC WRENCH |
| (r_{10}) RESISTANT SCREW HEAD | (r_{24}) USES SPECIAL TOOLS FOR TAMPER |
| (r_{11}) RETRACTIBILITY | |
| (r_{12}) RIVET TYPE | |
| (r_{13}) ROTARY MODE | |
| (r_{14}) SMALL HEAD CLEARANCE | |

20.5.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the fastener design domain is expressed in terms of the rules presented as follows:

(RULE 1) LOW WEIGHT-1

If the fastener is made of aluminum alloy

Then the fastener has LOW WEIGHT

(RULE 2) LOW WEIGHT-2

If the fastener is made of titanium
Then the fastener has LOW WEIGHT

(RULE 3) LOW WEIGHT-3

If the fastener is made of non-metallic material
Then the fastener has LOW WEIGHT

(RULE 4) AIRCRAFT

If the fastener has HIGH TENSILE STRENGTH & uses PNEUMATIC
WRENCH & has LOW WEIGHT
Then the fastener can be used in the AIRCRAFT industry

(RULE 5) SMALL HEAD CLEARANCE

If the fastener has internal wrenching hex socket
Then the fastener has SMALL HEAD CLEARANCE

(RULE 6) ROTARY MODE-1

If the fastener has coarse thread & Phillips driving recess
Then the fastener has ROTARY MODE

(RULE 7) ROTARY MODE-2

If the fastener has coarse thread & slot driving recess
Then the fastener has ROTARY MODE

(RULE 8) ROTARY MODE-3

If the fastener has coarse thread & hexagon driving recess
Then the fastener has ROTARY MODE

(RULE 9) CORROSION RESISTANCE-1

If the fastener is made of aluminum alloy
Then the fastener has CORROSION RESISTANCE

(RULE 10) CORROSION RESISTANCE-2

If the fastener is made of titanium
Then the fastener has CORROSION RESISTANCE

(RULE 11) CORROSION RESISTANCE-3

If the fastener is made of non-metallic material
Then the fastener has CORROSION RESISTANCE

(RULE 12) DIFFICULTY OF DISASSEMBLY

If the fastener is RIVET TYPE & has RESISTANT SCREW HEAD & uses
SPECIAL TOOLS FOR TAMPER
Then the fastener has DIFFICULTY OF DISASSEMBLY

(RULE 13) STANDARD DRIVING RECESS-1

If Phillips driving recess
Then the fastener has STANDARD DRIVING RECESS

(RULE 14) STANDARD DRIVING RECESS-2

If hexagon driving recess
Then the fastener has STANDARD DRIVING RECESS

(RULE 15) STANDARD DRIVING RECESS-3

If slot driving recess
Then the fastener has STANDARD DRIVING RECESS

(RULE 16) EASE OF DISASSEMBLY

If BOLT TYPE & ROTARY MODE & STANDARD DRIVING RECESS
Then the fastener has EASE OF DISASSEMBLY

(RULE 17) ELECTRICAL

If the fastener has ELECTRICAL AND THERMAL CONDUCTIVITY
Then the fastener is used in the ELECTRICAL industry

(RULE 18) ELECTRICAL AND THERMAL CONDUCTIVITY-1

If the fastener is made of copper alloy
Then the fastener has ELECTRICAL AND THERMAL CONDUCTIVITY

(RULE 19) ELECTRICAL AND THERMAL CONDUCTIVITY-2

If the fastener is made of titanium
Then the fastener has ELECTRICAL AND THERMAL CONDUCTIVITY

(RULE 20) ELECTRICAL AND THERMAL CONDUCTIVITY-3

If the fastener is made of stainless steel
Then the fastener has ELECTRICAL AND THERMAL CONDUCTIVITY

(RULE 21) ELECTRICAL AND THERMAL CONDUCTIVITY-4

If the fastener is made of non-metallic material
Then the fastener has ELECTRICAL AND THERMAL CONDUCTIVITY

(RULE 22) FATIGUE LOADING

If the fastener has thread that was rolled after heat treatment
Then the fastener is used for FATIGUE LOADING

(RULE 23) PNEUMATIC WRENCH

If the fastener has twelve point head bolt
Then the fastener uses PNEUMATIC WRENCH

(RULE 24) PRECISION

If the fastener has fine thread & Phillips head
Then the fastener has PRECISION

(RULE 25) RESISTANT SCREW HEAD

If the fastener has break away head
Then the fastener has RESISTANT SCREW HEAD

(RULE 26) RETRACTIBILITY

If the fastener has Phillips slot
Then the fastener has RETRACTIBILITY

(RULE 27) RIVET TYPE

If the fastener has chamfer point
Then the fastener is RIVET TYPE

(RULE 28) HIGH SHEAR STRENGTH

If the fastener is made of stainless steel
 Then the fastener has HIGH SHEAR STRENGTH

(RULE 29) STRENGTH TO WEIGHT RATIO-1

If the fastener is made of titanium
 Then the fastener has STRENGTH TO WEIGHT RATIO

(RULE 30) STRENGTH TO WEIGHT RATIO-2

If the fastener is made of aluminum
 Then the fastener has STRENGTH TO WEIGHT RATIO

(RULE 31) STRENGTH TO WEIGHT RATIO-3

If the fastener is made of alloy
 Then the fastener has STRENGTH TO WEIGHT RATIO

(RULE 32) HIGH OPERATION TEMPERATURE-1

If the fastener is made of titanium
 Then the fastener is used in HIGH OPERATION TEMPERATURE

(RULE 33) HIGH OPERATION TEMPERATURE-2

If the fastener is made of stainless steel
 Then the fastener is used in HIGH OPERATION TEMPERATURE

(RULE 34) HIGH OPERATION TEMPERATURE-3

If the fastener is made of non-metallic material
 Then the fastener is used in HIGH OPERATION TEMPERATURE

(RULE 35) HIGH TENSILE STRENGTH

If the fastener is made of stainless steel & has hexagon head
 Then the fastener has HIGH TENSILE STRENGTH

(RULE 36) SPECIAL TOOLS FOR TAMPER-1

If the fastener has plug in socket & 5 sided head
 Then the fastener needs SPECIAL TOOLS FOR TAMPER

(RULE 37) SPECIAL TOOLS FOR TAMPER-2

If the fastener has plug in socket & spanner head
 Then the fastener needs SPECIAL TOOLS FOR TAMPER

(RULE 38) VIBRATING ATMOSPHERE-1

If the fastener has nut & cotter pin & fine thread & washer
 Then the fastener is used in VIBRATING ATMOSPHERE

(RULE 39) VIBRATING ATMOSPHERE-2

If the fastener has nut & wire & fine thread & washer
 Then the fastener is used in VIBRATING ATMOSPHERE

20.5.3 FASTENER SYNTHESIS USING THE DESIGN SEARCH ALGORITHM (SEE CHAPTER 10.3)

Assume that the designer is faced with the problem of designing a fastener that is able to achieve the following specifications:

1. low weight (r_8);
2. usage in vibration atmosphere (r_{21});
3. fatigue loading (r_{17})

Table 20.6 shows the process states generated in the course of searching for a solution to the fastener design problem.

Table 20.6 The Design Search Algorithm Applied to the Fastener Design Problem

PROCESS STEP	Attributes Existing in OPEN	Attributes Existing in CLOSE	Candidate Unused Production Rules	Selected Production Rule (highest score)
1	r_8, r_{21}, r_{17}	\emptyset	1, 2, 3	1
2	m_4, r_{21}, r_{17}	r_8	38, 39	38
3	$m_4, m_{15}, m_9, m_{10}, m_{26}, r_{17}$	r_{21}, r_8	22	22
4	$m_4, m_{15}, m_9, m_{10}, m_{26}, m_{23}$ (consistent solution)	r_{17}, r_{21}, r_8	STOP	

20.6 FASTENER DESIGN EXAMPLE (CONTINUED)

Designing in established design domains may range from simple selection from a catalogue to composing systems from available components. The next example corresponds to a situation of choosing a design solution from a catalogue of existing fasteners.

20.6.1 THE SPECIFICATION AND DESIGN DESCRIPTION PROPERTIES

The Design Description Properties (identification of existing fasteners)

(m_1)	ball head machine screw	(m_{13})	flat socket head cap screw
(m_2)	binding head machine screw	(m_{14})	flat trim head machine screw
(m_3)	connector bolt	(m_{15})	flat undercut head machine screw
(m_4)	countersunk square neck bolt	(m_{16})	flat wood screw
(m_5)	countersunk square neck bolt	(m_{17})	hanger bolt
(m_6)	drilled eye bolt	(m_{18})	headless slotted set screw
(m_7)	elevator bolt	(m_{19})	hexagonal head bolt
(m_8)	fillister head machine screw	(m_{20})	hexagonal head machine screw
(m_9)	flat 100 degree head machine screw	(m_{21})	hexagonal head washer
(m_{10})	flat 82 degree head machine screw	(m_{22})	lag bolt
(m_{11})	flat fillister head machine screw	(m_{23})	oval head bolt
(m_{12})	flat head cap	(m_{24})	oval head machine screw
		(m_{25})	oval track

- | | |
|--|--|
| (m_{26}) oval trim head
machine screw | (m_{35}) socket type set
screw |
| (m_{27}) oval undercut head
machine screw | (m_{36}) square head bolt |
| (m_{28}) oval wood screw | (m_{37}) square head set
screw |
| (m_{29}) pan head machine
screw | (m_{38}) step bolt |
| (m_{30}) round head machine
screw | (m_{39}) t-bolt |
| (m_{31}) round head square
neck carriage bolt | (m_{41}) truss head machine
screw |
| (m_{32}) round washer head
machine screw | (m_{42}) twelve point head
bolt |
| (m_{33}) round wood screw | (m_{43}) wedge bolt |
| (m_{34}) socket head cap
screw | (m_{44}) weld screw |

The Specification Properties

- | | |
|--|---|
| (r_1) USED FOR ADDED
ATTRACTIVENESS | (r_6) COUNTER-BORED HOLES
ATTACHMENT |
| (r_2) USED IN AIRCRAFT
INDUSTRY | (r_7) COVERING LARGE
DIAMETER HOLES |
| (r_3) AMPLE BEARING SURFACE | (r_8) EXTERNAL WRENCHING |
| (r_4) CLOSER TOLERANCE | (r_9) FARM MACHINERY
ATTACHMENT |
| (r_5) CONTACT ATTACHMENT (in
the electrical industry) | (r_{10}) FLAT SLOTTED HEAD |
| | (r_{11}) FLUSH SCREW |

- (*r*₁₂) FLUSH SURFACE ATTACHMENT
- (*r*₁₃) FLUSH WOOD WORK ATTACHMENT
- (*r*₁₄) FREQUENT DISASSEMBLY
- (*r*₁₅) GENERAL PURPOSE SERVICE
- (*r*₁₆) GENEROUS BEARING SURFACE ATTACHMENT
- (*r*₁₇) GOOD WOOD WORK APPEARANCE
- (*r*₁₈) USED FOR HANGING MARBLE FASCIA
- (*r*₁₉) HEAVY CONSTRUCTION EQUIPMENT ATTACHMENT
- (*r*₂₀) HEAVY DUTY APPLICATIONS
- (*r*₂₁) HIGH STRENGTH
- (*r*₂₂) HIGH WRENCHING TORQUE
- (*r*₂₃) INTERNAL WRENCHING
- (*r*₂₄) LARGE BEARING SURFACE ATTACHMENT
- (*r*₂₅) USED IN MACHINE TOOL INDUSTRY
- (*r*₂₆) USED IN MASONRY WORK
- (*r*₂₇) MAXIMUM DRIVING POWER
- (*r*₂₈) METAL PARTS ATTACHMENT
- (*r*₂₉) METAL-TO-WOOD ATTACHMENT
- (*r*₃₀) RAIL ATTACHMENT
- (*r*₃₁) RESILIENT MATERIAL ATTACHMENT
- (*r*₃₂) SHEET METAL ATTACHMENT
- (*r*₃₃) SHEET METAL TO STEEL ATTACHMENT
- (*r*₃₄) SHORT SCREW
- (*r*₃₅) SPECIAL PURPOSE MACHINE SCREW
- (*r*₃₆) TAPPED AND COUNTER-BORED HOLES
- (*r*₃₇) THIN METALS AND PLASTICS ATTACHMENT
- (*r*₃₈) TURNING PREVENTING
- (*r*₃₉) VALVE ATTACHMENT
- (*r*₄₀) WIDE AND THIN BEARING SURFACE ATTACHMENT
- (*r*₄₁) WOOD ATTACHMENT
- (*r*₄₂) WRENCH DISFIGUREMENT PROTECTION

20.6.2 THE PRODUCTION RULES

A portion of the domain-specific knowledge relevant to the fastener catalogue domain is expressed in terms of the following production rules:

(RULE 1) SPECIAL PURPOSE MACHINE SCREW

If ball head machine screw
Then the fastener is used as SPECIAL PURPOSE MACHINE SCREW

(RULE 2) CONTACT ATTACHMENT

If connector bolt
Then the fastener is used for CONTACT ATTACHMENT in the electrical industry

(RULE 3) FARM MACHINERY ATTACHMENT

If countersunk square neck bolt
Then the fastener is used for FARM MACHINERY ATTACHMENT

(RULE 4) HEAVY CONSTRUCTION EQUIPMENT ATTACHMENT

If countersunk square neck bolt
Then the fastener is used for HEAVY CONSTRUCTION EQUIPMENT ATTACHMENT

(RULE 5) VALVE ATTACHMENT

If drilled eye bolt
Then the fastener is used for VALVE ATTACHMENT

(RULE 6) CLOSER TOLERANCE

If drilled eye bolt
Then the fastener has CLOSER TOLERANCE

(RULE 7) HIGH STRENGTH

If drilled eye bolt
Then the fastener has HIGH STRENGTH

(RULE 8) WIDE AND THIN BEARING SURFACE

If elevator bolt
Then the fastener is used for WIDE AND THIN BEARING SURFACE ATTACHMENT

Cause: an elevator bolt has a thin circular head with extra large diameter, and a square shoulder under the head that prevents turning when a nut is applied.

(RULE 9) TURNING PREVENTING

If elevator bolt
Then the fastener has TURNING PREVENTING

Cause: an elevator bolt has a square shoulder under the head that prevents turning when a nut is applied.

(RULE 10) COUNTER-BORED HOLES ATTACHMENT-1

If fillister head machine screw
Then the fastener is used for COUNTER-BORED HOLES ATTACHMENT

Cause: a fillister head machine screw has a small diameter but higher diameter than a round head machine screw. It has a deep slot when it is slotted.

(RULE 11) COUNTER-BORED HOLES ATTACHMENT-2

If flat fillister head machine screw
Then the fastener is used for COUNTER-BORED HOLES ATTACHMENT

(RULE 12) THIN METALS AND PLASTICS ATTACHMENT

If flat 100 degree head machine screw
Then the fastener is used for THIN METALS AND PLASTICS ATTACHMENT

Cause: flat 100 degree head machine screw has larger head than 82 degree design and are available with a slotted or Phillips driving recess.

(RULE 13) FLUSH SCREW

If flat fillister head machine screw
Then the fastener is FLUSH SCREW

Cause: a flat fillister head machine screw is similar to a standard fillister but without the oval top. It is available with a slot drive recess only as a machine screw.

(RULE 14) FLAT SLOTTED HEAD

If flat head cap
Then the fastener has a FLAT SLOTTED HEAD

Cause: a flat head cap is a flat countersunk bolt that often has a slotted head.

(RULE 15) INTERNAL WRENCHING

If flat socket head cap screw
Then the fastener is used for INTERNAL WRENCHING

Cause: a flat socket head cap screw is a flat countersunk screw or bolt with hexagonal socket or recess.

(RULE 16) SHORT SCREW

If flat undercut head machine screw
Then the fastener is SHORT SCREW

(RULE 17) FLUSH WOOD WORK ATTACHMENT-1

If flat wood screw
Then the fastener is used for FLUSH WOOD WORK ATTACHMENT

Cause: a flat wood screw has a countersunk flat head, and a slotted or Phillips driving recess.

(RULE 18) FLUSH WOOD WORK ATTACHMENT-2

If hanger bolt
Then the fastener is used for FLUSH WOOD WORK ATTACHMENT

Cause: a hanger bolt has a gimlet point on one end and a machine screw thread on the other end.

(RULE 19) FLUSH SURFACE ATTACHMENT-1

If flat 82 degree head machine screw
Then the fastener is used for FLUSH SURFACE ATTACHMENT

Cause: a flat 82 degree head machine screw has a countersunk section aids centering, and it is available with slotted, clutch, Phillips or hexagon-socket driving recess.

(RULE 20) FLUSH SURFACE ATTACHMENT-2

If flat trim head machine screw
Then the fastener is used for FLUSH SURFACE ATTACHMENT

Cause: a flat trim head machine screw is similar to an 82 degree flat head machine screw except for the reduced depth of the countersink. It has only a Phillips driving recess.

(RULE 21) FLUSH SURFACE ATTACHMENT-3

If flat undercut head machine screw
Then the fastener is used for FLUSH SURFACE ATTACHMENT

Cause: a flat undercut head machine screw has a standard 82 degree flat head with lower third of the countersink removed.

(RULE 22) FLUSH SURFACE ATTACHMENT-4

If headless slotted set screw
Then the fastener is used for FLUSH SURFACE ATTACHMENT

Cause: a headless slotted set screw can be driven (with a screw driver) below the work surface.

(RULE 23) AMPLE BEARING SURFACE

If hexagonal head bolt
Then the fastener has AMPLE BEARING SURFACE

(RULE 24) EXTERNAL WRENCHING

If hexagonal head machine screw
Then the fastener is used for EXTERNAL WRENCHING

Cause: a hexagonal head machine screw is similar to a hexagonal head bolt but smaller.

(RULE 25) WRENCH DISFIGUREMENT PROTECTION

If hexagonal head washer
Then the fastener is used for WRENCH DISFIGUREMENT PROTECTION

Cause: a hexagonal head washer has a hexagonal head (or a slotted head) with a washer section at base in order to protect work surface against wrench disfigurement.

(RULE 26) MASONRY WORK

If lag bolt
Then the fastener is used for MASONRY WORK

(RULE 27) HEAVY DUTY APPLICATIONS

If oval head bolt
Then the fastener is used for HEAVY DUTY APPLICATIONS

Cause: a oval head bolt has a slotted oval head for flush mounted seating. It is similar to a machine screw but designed for heavier duty applications.

(RULE 28) RAIL ATTACHMENT

If oval track
Then the screw is used for RAIL ATTACHMENT

Cause: an OVAL TRACK fastener joins rails in railroads and electric railways.

(RULE 29) ADDED ATTRACTIVENESS SCREW-1

If oval head machine screw
Then the screw has ADDED ATTRACTIVENESS

Cause: an oval head machine screw is similar to a standard flat head, but has added attractiveness because of the rounded outer surface.

(RULE 30) ADDED ATTRACTIVENESS SCREW-2

If oval trim head machine screw
Then the screw has ADDED ATTRACTIVENESS

Cause: an oval trim head machine screw is similar to standard oval head, except for the reduced countersink depth. It has only a Phillips driving recess.

(RULE 31) ADDED ATTRACTIVENESS SCREW-3

If oval undercut head machine screw
Then the screw has ADDED ATTRACTIVENESS

Cause: an oval undercut head machine screw is similar to an oval head with the lower third of the countersink removed. It has a slotted or Phillips driving recess.

(RULE 32) GOOD WOOD WORK APPEARANCE

If oval wood screw
Then the screw provides GOOD WOOD WORK APPEARANCE

Cause: an oval wood screw has a countersunk oval head with rounded protrusion. It has a slotted or Phillips driving recess.

(RULE 33) MAXIMUM DRIVING POWER-1

If binding head machine screw
Then the fastener has MAXIMUM DRIVING POWER

Cause: a binding head machine screw is similar to a pan head machine screw. The fastener is supplied undercut when specified, for binding and eliminating fraying of wire in electrical work. It has slotted or Phillips driving recess.

(RULE 34) MAXIMUM DRIVING POWER-2

If pan head machine screw
Then the fastener has MAXIMUM DRIVING POWER

Cause: a pan head machine screw has a large diameter and high outer edges. It has a slotted or Phillips driving recess.

(RULE 35) GENERAL PURPOSE SERVICE

If round head machine screw
Then the fastener is used for GENERAL PURPOSE SERVICE

Cause: a round head machine screw has a large slot depth and ample bearing surface (not as much as a pan head screw). It has a slotted or Phillips driving recess.

(RULE 36) LARGE BEARING SURFACE ATTACHMENT

If round washer head machine screw
Then the fastener is used for LARGE BEARING SURFACE ATTACHMENT

Cause: a round washer head machine screw has a round head with integral washer and a slotted or Phillips driving recess.

(RULE 37) WOOD ATTACHMENT-1

If lag bolt
Then the fastener is used for WOOD ATTACHMENT

Cause: a lag bolt has a hexagonal or square head and 60 degree gimlet conical point.

(RULE 38) WOOD ATTACHMENT-2

If round wood screw
Then the fastener is used for WOOD ATTACHMENT

Cause: a round wood screw has a round head that protrudes above the work. It has either a slotted or Phillips driving recess.

(RULE 39) WOOD ATTACHMENT-3

If round head square neck carriage bolt

Then the fastener is used for WOOD ATTACHMENT

(RULE 40) METAL-TO-WOOD ATTACHMENT

If round wood screw

Then the fastener is used for METAL-TO-WOOD ATTACHMENT

(RULE 41) TAPPED AND COUNTER-BORED HOLES

If socket head cap screw

Then the fastener is used for TAPPED AND COUNTER-BORED HOLES

Cause: socket head cap screw has hexagonal socket or recess.

(RULE 42) INTERNAL WRENCHING-1

If flat socket head cap screw

Then the fastener is used for INTERNAL WRENCHING

Cause: a flat socket head cap screw is a flat countersunk screw or bolt with hexagonal socket or recess.

(RULE 43) INTERNAL WRENCHING-2

If socket type set screw

Then the fastener is used for INTERNAL WRENCHING

(RULE 44) GENEROUS BEARING SURFACE ATTACHMENT

If square head bolt

Then the fastener is used for GENEROUS BEARING SURFACE ATTACHMENT

Cause: a square head bolt has a square head which is good for wrench tightening.

(RULE 45) HIGH WRENCHING TORQUE

If square head set screw

Then the fastener is used for HIGH WRENCHING TORQUE application

Cause: the square head has sharp, well-defined corners and threads extending to it.

(RULE 46) RESILIENT MATERIAL ATTACHMENT-1

If step bolt

Then the fastener is used for RESILIENT MATERIAL ATTACHMENT

Cause: a step bolt has a square neck that prevents turning.

(RULE 47) RESILIENT MATERIAL ATTACHMENT-2

If round head square neck carriage bolt
Then the fastener is used for RESILIENT MATERIAL ATTACHMENT

(RULE 48) SHEET METAL ATTACHMENT

If round head square neck carriage bolt
Then the fastener is used for SHEET METAL ATTACHMENT

(RULE 49) SHEET METAL - TO - STEEL ATTACHMENT

If step bolt
Then the fastener is used for SHEET METAL TO STEEL ATTACHMENT

(RULE 50) MACHINE TOOL INDUSTRY

If t-bolt
Then the fastener is used in MACHINE TOOL INDUSTRY

(RULE 51) FREQUENT DISASSEMBLY-1

If socket type set screw
Then the fastener enables FREQUENT DISASSEMBLY

(RULE 52) FREQUENT DISASSEMBLY-2

If thumb screw
Then the fastener enables FREQUENT DISASSEMBLY

Cause: a thumb screw has a standard thread thumb screw that provides large thumb gripping spade.

(RULE 53) COVERING LARGE DIAMETER HOLES

If truss head machine screw
Then the fastener is used for COVERING LARGE DIAMETER HOLES

Cause: a truss head machine screw is similar to a round head machine screw except for the shallower head and larger diameter. It has a slotted, clutch or Phillips head.

(RULE 54) AIRCRAFT INDUSTRY

If twelve point head bolt

Then the fastener is used in AIRCRAFT INDUSTRY

Cause: a twelve point head bolt has a double hexagon head and therefor has a superior adaptability to pneumatic wrenching.

(RULE 55) HANGING MARBLE FASCIA

If wedge bolt

Then the fastener is used for HANGING MARBLE FASCIA on buildings

(RULE 56) METAL PARTS ATTACHMENT

If weld screw

Then the fastener is used for METAL PARTS ATTACHMENT

Cause: a weld screw has lugs or weld projections on top or underside of its head to facilitate attachment to metal parts by resistance welding.

APPENDIX A - AUTOMOBILE DESIGN

In this appendix, technical and operational details related to the design of the main parts of automobiles and their components are provided. The production rules presented in Section 1.2 rely very heavily on these domain-specific knowledge.

A.1 ENGINES

A variety of engine types have been used in foreign and domestic vehicles. They are generally *internal-combustion engines* that burn gasoline or diesel fuel oil. Internal-combustion engines can be classified in several ways. Of the many possibilities, most passenger-car engines run on gasoline, have spark ignition, and are of the liquid-cooled, four-stroke-cycle, overhead-valve, carbureted, reciprocating type.

Diesel engines, which burn fuel oil, are becoming more common in automobiles. Diesel engines inject fuel into the cylinders, where it is ignited by the heat of compression; they do not require spark *ignition systems*.

Most engines have carbureted fuel systems; these mix gasoline with air in the *carburetor* to form a combustible mixture. All diesel engines and some gasoline engines, however, use fuel injection instead of a carburetor. In the diesel engine, the fuel is injected directly into the engine cylinders.

Almost all automobiles use reciprocating engines, which have pistons that move up and down in cylinders. The crankshaft, the main shaft of the engine, converts the reciprocating motion of the pistons into rotary motion. Some, however, use *Wankle engines*, which are a rotary type of engine.

A.1.1 Spark-Ignition Engine

HISTORY

The invention and early development of internal-combustion engines is usually credited to three Germans. Nikolaus Otto patented and built (1876) the first such engine; Karl Benz built the first automobile to be powered by such an engine (1885); and Gottlieb Daimler designed the first high-speed internal-combustion engine (1885) and carburetor. Rudolf Diesel invented a successful compression-ignition engine (the diesel engine) in 1892.

OPERATION

The operation of the internal-combustion reciprocating engine employs either a four-stroke cycle or a two-stroke cycle (a stroke is one continuous movement of the piston within the cylinder.) In the four-stroke cycle, also known as the Otto cycle, the downward movement of a piston located within a cylinder creates a partial vacuum. Two valves located inside the combustion chamber are controlled by the motion of a camshaft connected to the crankshaft. The four strokes are called, in order of sequence, intake, compression, power, and exhaust. On the first stroke, the intake valve is opened while the exhaust valve is closed; atmospheric pressure forces a mixture of gas and air to fill the chamber. On the second stroke, the intake and exhaust valves are both closed as the piston starts upward.

The mixture is compressed from normal atmospheric pressure (1 kg/sq. cm, or 14.7 lb./sq. in) to between 4.9 and 8.8 kg/sq. cm (70 and 125 lb./sq. in). During the third stroke, the compressed mixture is ignited -- either by compression ignition or by spark ignition. The heat produced by the combustion causes the gases to expand within the cylinder, thus forcing the piston downward. The piston's connecting rod transmits the power from the piston to the crankshaft. This assembly changes reciprocating -- in other words, up-and-down or back-and-forth motion -- to rotary motion. On the fourth stroke, the exhaust valve is opened so that the burned gases can escape as the piston moves upward; this prepares the cylinder for another cycle.

Internal-combustion spark-ignition engines with two-stroke cycles combine intake and compression in a single first stroke, and power and exhaust in a second stroke.

A.1.2 Starting System

Internal-combustion engines require some type of starting system. Larger engines may use compressed air or an electric starting system. The latter includes a starter --a high-torque electric motor -- to turn the crankshaft until the engine starts. Starting motors are extremely powerful for their size and are designed to utilize high currents (200 to 300 amperes). The large starting currents can cause a battery to drain rapidly; for this reason a heavy-duty battery is usually used. Interrupting this connection is an

electrical switch called a solenoid, which is activated by the low-voltage starting switch. In this way the ignition switch can be located away from the starter and yet still turn the starter on and off.

A.1.3 Diesel Engine

A diesel engine is a type of internal-combustion engine that is similar to the gasoline engine, but requires no electrical ignition system or carburetor. It was invented by Rudolf Diesel, a German engineer, who obtained a patent for the design in 1892.

Diesels employ high compression ratios to elevate the compressed air temperature sufficiently to ignite a low-grade fuel that is injected into the cylinder. Components of diesels are usually heavier than those of gasoline engines because of the additional structural strength needed to obtain the higher compression ratio and power output.

Diesel engines employ a system of fuel injection to spray the fuel into the cylinder after the air has been compressed by the piston. This mixture burns; the expanding gases push the piston down and thus supply power. The timing of this fuel injection is just as critical as is the spark that ignites the fuel in the gasoline engine. Therefore, the injection mechanisms are mechanically linked to the crankshaft. Since each cylinder takes in and compresses a fixed amount of air, the power of the engine is varied by the amount of fuel injected. The timing, as well as the duration and pressure of fuel injection, is designed so that the maximum useful energy is obtained from the fuel for a particular range of speed, power, acceleration, or other working conditions.

Diesel engines, like other internal-combustion engines, require an exhaust system, a cooling system, and a starting system. Because of the unusually high compression ratios, diesel engines need a powerful starting system. Some diesel engines use an electric motor or an auxiliary gasoline engine, whereas others use compressed air or spark ignition to start the engine. Diesel engines have always been popular power plants for large vehicles such as buses, trucks, locomotives, and ships. Small diesels have been used in automobiles, although the noise, soot, and pollutants they produce have discouraged that use in the United States.

The diesel has been successful because of its operating advantages, such as low maintenance costs, greater efficiency, high power output, and fuel economy under all loads.

A.1.4 Power Capacity

The power capacity of an engine depends on a number of characteristics, including the volume of the combustion chamber. The volume can be increased by increasing the size of the piston and cylinder and by increasing the number of cylinders.

A.1.5 Engine Location

In rear-drive cars with front engines, one drive shaft is needed. In front-drive cars with front-mounted engines, two are needed; one for each front wheel. Although most conventional American cars use a front-mounted engine to drive the rear wheels, an increasing number of newer models use front-wheel drive; where the weight of the engine on the front wheels provides better traction on slippery roads. In some cars designed particularly for bad weather and rough roads, the engine drives all four wheels.

A.1.6 The Fuel System

An automotive fuel system consists of the carburetor or fuel injector, the fuel tank, the fuel pump, and the fuel filter, along with tubing connecting the parts.

CARBURETOR

A carburetor is a device that vaporizes a liquid fuel such as gasoline and mixes it with air in the proper ratio for combustion in an internal-combustion engine, such as the gasoline engine that powers most automobiles. Under ordinary conditions the gasoline to air weight ratio should be about 1:15 (1 part gasoline to 15 parts air). A higher ratio of gasoline is called a 'richer' mixture, and a lower ratio is called 'leaner'.

A simple form of carburetor consists of a float chamber, a jet nozzle, and an air chamber that is narrowed at one point. Such a narrowing in a chamber or tube is called a venturi. A float valve keeps the gasoline at a constant level in the float chamber. When the engine is running, the motion of the pistons creates a vacuum, drawing air into the air chamber, where it is accelerated by the venturi. In accordance with Bernoulli's law, this high-velocity air creates a low-pressure region, and the jet nozzle, which is attached in this region, draws a fine spray of gasoline from the float chamber into the venturi. Here it mixes with the air, in a manner much like that of a perfume atomizer. The mixture of gasoline vapor and air is then fed to the engine cylinders, where it is ignited. A throttle valve, which is actuated by the accelerator pedal, governs engine speed by regulating the amount of the gasoline-air mixture that enters the engine. A choke valve at the entrance to the carburetor is used to reduce the amount of air entering the chamber when the engine is cold. Less air means a richer mixture that can be more easily ignited by the spark plugs. As the engine warms up, the choke valve gradually opens, reducing the richness of the mixture.

FUEL INJECTOR

In practice, carburetors use various means to ensure an optimal mixture of gasoline and air under differing conditions, including idling and rapid acceleration. Instead of having a carburetor, an engine can have a system of fuel injection, which delivers a

metered quantity of gasoline directly to each cylinder. Fuel injection has always been used with diesel engines; it has also been gaining in use with gasoline engines.

IMPROVED FUEL ECONOMY

Recent advances in improved fuel economy include aerodynamic drag reduction, demand-actuated fan drive systems, steel radial tires, synthetic lubricants, and engine and drive train modifications.

Drag can be dramatically reduced by an air deflector. This device, which comes in various shapes, improves the streamlining by deflecting air over the top of the trailer.

The new fan systems -- used also in some automobiles -- turn the fan on and off as needed, either by a temperature-sensing device or by a centrifugal clutch that disconnects the fan when the engine, and hence vehicle, speed is high enough to provide adequate cooling. Having the fan disconnected when not needed can save a considerable amount of energy.

Other means to improve fuel economy include turbochargers that use waste energy to improve engine efficiency, advanced diesel engines, sleeker, aerodynamic designs, and the increased use of plastics and lightweight materials.

A.1.7 Cooling System

An automotive engine requires cooling because the burning fuel produces much more heat than the engine can convert into useful power. The thermal efficiency (the percentage of energy actually converted into power) is somewhere between 20 and 25 percent for an internal-combustion engine. The 75 to 80 percent of the energy not converted into power must be discarded by the engine. Some is lost in the hot exhaust gases leaving the engine; some is removed by the engine lubrication system; and some is removed by the engine cooling system. A liquid cooling system typically removes about 30 to 35 percent of the heat produced by the engine.

Two types of cooling systems are used: *air cooling* and *liquid cooling*. An air-cooled engine has separate cylinders and cooling fins both on the cylinders and the cylinder heads. The liquid-cooled engine, which is used in most automobiles, circulates a cooling liquid, or coolant, between the radiator and the water jackets surrounding the cylinders and combustion chambers. The coolant is a mixture of water and antifreeze. The primary function of the cooling system is to prevent the engine from overheating after it reaches operating temperature; a related function is to speed the warm-up of a cold engine.

The purpose of the quick warm-up is to take the engine out of its most inefficient and wear-causing mode of operation as quickly as possible. A cold engine requires a rich mixture (excess of gasoline); thus the exhaust gas has more atmospheric pollutants. Because the lubricating oil is still cold, it is viscous and flows slowly; therefore, the moving metal parts wear more rapidly.

The cooling system is monitored and controlled by a thermostat located in the

upper hose line. When the engine is cold, the thermostat is in the closed position, blocking off this line so that no coolant can flow to the radiator. The retained heat quickly causes the engine temperature to rise. As the temperature reaches the optimum value, the thermostat opens, and hot coolant begins to flow to the upper part of the radiator.

AIR-COOLED ENGINES

Air-cooled engines are popularly used to power small cars, often require no moving parts, and therefore little or no maintenance, for the cooling system. The head, or uppermost part, of the cylinder and the cylinder block have fins cast into them; these fins increase the surface exposed to the surrounding air, allowing more heat to be radiated. Usually a cover or shroud channels the air flow over the fins. A fan is sometimes included if the engine is located away from a stream of fast-moving air.

WATER-COOLED ENGINES

Water-cooled engines have water jackets built into the engine block. These jackets surround the cylinders. Usually a centrifugal water pump is used to circulate the water continuously through the water jackets. In this way the high heat of combustion is drawn off the cylinder wall into the circulating water. The water must then be cooled in a radiator that transfers the heat energy of the water to the radiator's cooler surrounding fluid. The surrounding fluid can be air or water, depending on the application of the engine.

Radiator

The function of the radiator is to cool the circulating coolant. The radiator contains a series of coolant passages surrounded by a series of air passages that permit air to flow from front to back through the radiator. The heat in the coolant passes to the radiator metal and then to the air circulating through the radiator. Thus, the coolant is cooled. From the lower end of the radiator, it flows to the engine water jackets. Circulation is maintained by the water pump.

Water Pump and Fan

The water pump is an impeller-type pump, driven by a V-belt from the crankshaft pulley. The engine fan is mounted on the water-pump pulley. Its function is to maintain the flow of air through the radiator. The forward motion of the car also helps this air movement. Some vehicles have a clutch drive between the fan hub and the pulley. This device reduces fan speed by thermostatic means when full fan speed and air circulation are not needed. Power consumption and noise are thus reduced.

Sealed Systems

Cooling systems on most vehicles today are sealed. This allows them to operate at pressures as much as 1.0 N/sq. m (15 lb./sq. in) over atmospheric pressure, raising the boiling point of the coolant to as much as 126.6 deg C (260 deg F). The system can therefore operate at a higher temperature, with greater efficiency. The pressure buildup is made possible by the radiator pressure cap, which contains a pressure valve and a vacuum valve. If the pressure rises above a specified value, the pressure valve opens. When the engine cools off and the coolant contracts, the vacuum valve admits air or coolant from the expansion tank.

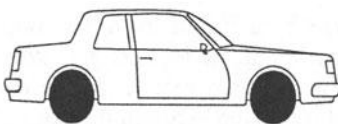
Expansion Tank

Most modern automobiles have an expansion tank in the cooling system. The expansion tank is a container for coolant to flow into when the engine heats up and the coolant expands. When the engine cools, coolant can flow from the tank back into the cooling system. This arrangement reduces the loss of coolant.

A.2 THE BODY

A.2.1 Body Design

There are many vehicle body designs such as (see figure A.1 below): (1) A *sedan* has a closed body design with two or four doors, two cross seats, and usually accommodates five or six people. In the two-door model, the backs of the front seats tip forward to give access to the rear seat; (2) A *hatchback*, although similar to a sedan, enables access from the rear; (3) A *convertible* has an open-body design and is similar to the two-door sedan, but has a folding top that can be raised or lowered. A hardtop is similar to the two- or four-door sedan except that it has no side members between the front and rear windows; (4) A *station wagon* has a special body available on a more or less standard chassis. It has cross seats in the front and either cross or side seats in the rear. It may be built to accommodate up to nine people, and generally also includes additional luggage or cargo space; (5) A *sports car* is a low, comparatively small car. It usually seats two, and is designed for speed and maneuverability; (6) A *tow* is a strong off-road vehicle; (7) An *economy car* is a smaller and simpler version of the sedan. Generally, it is designed to economize in terms of costs and space.



A Sedan (2 Door)



A Hatchback (4 Door)

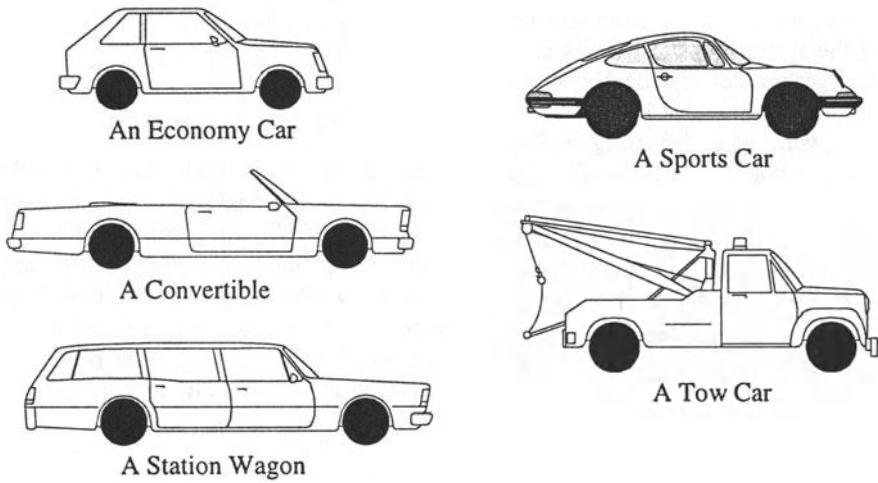


Figure A.1 Seven Common Vehicle Body Designs

A.2.2 Frames

Two types of frames, full and stub, are used. The *full frame* has side, front, back, and cross members welded into a single assembly that supports all other parts of the vehicle. The *stub frame* has separate frames for the front and back that are welded to the car body, with the body forming the center and connecting support.

A.3 STEERING SYSTEM

An automobile steering system allows the driver to control the vehicle's direction. Rotation of the steering wheel is translated, through gearing and a network of rods and joints (the steering linkage), into right or left movement of the car's front wheels.

There are two basic types of steering systems: rack-and-pinion, and worm-and-roller (or worm-and-key):

- *Rack-and-pinion* steering is the simplest, and most commonly used system in modern cars. The steering shaft, which is connected at one end to the steering wheel, has a pinion gear at its opposite end. The gear meshes with threads in a steering rack, which is mounted across the car, and is connected by tie rods to the front wheels. The steering wheel turns the steering shaft, and the pinion gear on the shaft's end moves the steering rack to the right or left. Thus the wheels move. Rack-and-pinion systems use few moving parts but perform precisely and efficiently.
- In *worm-and-roller* systems the steering shaft ends in a gearbox, which is connected to the steering linkage by a special rod called a Pitman arm. On the left side of the car, an idler arm is mounted parallel to the right-mounted Pitman arm. The arms are connected by a relay rod. Inserted into the relay rod are tie rods connecting it to the two front wheels, and short tire-steering arms that are joined to

the tie rods by ball joints. Inside the gearbox the worm (a spiral-threaded gear on the end of the steering shaft) meshes at right angles with a roller or key (a roller is a threaded gear; a key is a device with a single tooth that engages the threads of the worm gear.) As the steering wheel is turned, the roller or key moves right or left along the worm gear; swiveling the Pitman arm.

Many modern vehicles have *power-assisted steering*, which uses hydraulic pressure to reduce the effort needed to steer. In the most common system, a power steering pump (connected by a belt to the engine) sends high-pressure fluid to the steering gearbox, where it aids in moving the gears. Innovative *4-wheel steering* (4WS) is now available in two modes: either all 4 wheels turn in the same direction, or front and rear wheels turn in opposite directions. Both systems are said to produce the same effects: parking is far easier, and high-speed maneuvers are safer because "fishtailing" is reduced. In either mode, 4WS is still relatively untried, but it may prove eventually to be the steering system of the future.

A.4 SUSPENSION SYSTEM

An automotive suspension system supports the frame and body of a vehicle, attaching them to the wheels. Its main purpose is to absorb road shocks caused by irregularities in the road surface in order to reduce the bumps transmitted to the occupants. The system also improves safety and road-handling ability by maintaining better contact between the wheels and the road.

Most modern motor vehicles have *separate springs* at each wheel, with supports arranged in several ways. There is a *shock absorber* at each wheel to help smooth out spring action. Different arrangements are often used for the suspension at the front and rear of the vehicle, because the front wheels must pivot from side to side for steering as well as up and down to absorb road shocks. Some cars use a *torsion bar*, which is a special type of spring. *Stabilizers* are sometimes used to prevent excessive lean-out on turns.

A.4.1 Springs

Many suspension systems use coil, or helical, springs. Rear suspensions often use leaf springs. A leaf spring is essentially a flat bar that resists motion. The two ends of the leaf spring are attached to the frame, and the center is attached to the axle housing. This enables the wheel and housing to move up and down with respect to the frame when the car hits a bump in the road.

A.4.2 Torsion Bars

Some vehicles use torsion bars instead of coil springs at certain points in the system. The torsion bar twists as road irregularities are encountered. Its front end is attached to a lower control arm, and its rear end is attached to the car frame. As the lower

control arm pivots up and down, the torsion bar twists and behaves more or less like a coil spring.

A.4.3 Shock Absorbers

A shock absorber offers a large resistance to movement and can quickly absorb vibrations. Its purpose in a suspension system is to moderate wheel movement and to prevent continuing vibrations after a wheel has passed over a bump. A spring without a shock absorber would continue to expand and contract for a long time; thus, causing poor car control and a rough ride. On many front suspension systems the shock absorber is attached at the top to the frame and at the bottom to the lower control arm. In rear suspensions it is attached in a number of different ways.

A.4.4 Stabilizers

Some front suspension systems also use stabilizers. The stabilizer is a long steel rod that is attached at each end to the two lower control arms. Its purpose is to prevent excessive lean-out on turns. When the vehicle goes around a curve, centrifugal force tends to make the car lean out. This action puts more weight on the outer front spring, which thus is compressed more than the inner spring on the turn. As the outer spring is compressed, it imparts a twist to the stabilizer bar. The inner spring on the turn tends to expand, because it has less weight on it. This twists the stabilizer bar in the opposite direction. The stabilizer bar resists this twist and thus reduces the amount that the vehicle can lean out on a turn.

A.4.5 Automatic Level Control

Some cars have a system called automatic level control, which compensates for any change in loading of the rear of the car. The system includes a compressor to supply compressed air, two special shock absorbers with air domes, and a height-control valve. When the weight in the rear is increased (by passengers or cargo), the rear settles. This triggers the height-control valve, which sends compressed air from the pump through the valve into the air domes in the shock absorbers. The compressed air causes the shock absorbers to extend and thus raise the rear of the car. When the original height is reached, the valve shuts off the flow of compressed air. When the rear of the car is unloaded, it will rise above its original height. When this happens, the height-control valve opens to release some of the air from the shock-absorber air domes, causing the rear end to settle to the original height.

A.5. THE BRAKE SYSTEM

A.5.1 Drum Brake

A drum brake has two convex brake shoes that press against a rotating brake drum. Automobile drum brakes have internal brake shoes that are inside a tightly closed drum and are actuated by hydraulic pressure. Such a brake has the advantage that it can keep out water and dust effectively; its ability to dissipate heat is limited, however. When drum brakes heat up, they are subject to fading--a decrease in braking effectiveness during extended use of the brake. This happens because the increased temperature causes a decrease in the friction of the brake-lining material. Special materials have been developed that can be used for the linings to improve resistance to fading.

A.5.2 Disk Brake

The disk brake has a block that presses against the flat surface of a disk rather than against the rim. An example is the caliper disk brake, which was originally developed for aircraft and is now increasingly being used in automobiles; the same type is used in bicycles. Two opposed blocks (brake shoes) squeeze the disk between them like a pair of calipers. Disk brakes dissipate heat rapidly because they are not enclosed, thus allowing air to flow over them and carry away heat.

Newer, heavier cars are often equipped with disk brakes, especially on the front wheels. Rear-wheel braking is not as critical, because the weight of the vehicle is shifted to the front wheel during braking, thereby reducing the braking force needed at the rear wheels.

A.5.3 Hydraulic Brakes

Until about 1930, automobiles were braked mechanically. Such a system, however, made it difficult to brake all the wheels equally. The increased weight and speed of vehicles also required that the driver exert a greater pedal pressure. Both these problems were solved with the development of modern hydraulic braking systems.

In a hydraulic system, depression of the brake pedal moves a piston in a master cylinder, forcing hydraulic fluid through piping to a cylinder at each wheel. Called wheel cylinders or slave cylinders, these are each fitted with pistons moved by the pressure of the fluid, which brings the brake lining into contact with the rotating brake drum or disk, producing a braking force. The hydraulic system can be designed to multiply the force transmitted from the pedal to the wheel cylinder, which reduces the pedal pressure that the driver must apply.

A.5.4 Power Brakes

Even with the force-multiplying feature of hydraulic brakes, as vehicles became heavier and faster the pedal pressure required to brake the vehicle increased beyond a comfortable, safe level. Power brakes were developed to solve this problem. In automobiles they use the vacuum created by the engine during its intake stroke to increase the pressure applied to the piston in the master cylinder, reducing the required pedal pressure. If the power-assisting mechanism should fail, or if the engine stalls, the brakes will not fail completely, although greater pedal pressure will be needed.

Power-assisted brake systems are also needed in such heavy vehicles as buses, trucks, and railroad trains. One such system is the pneumatically operated Westinghouse air brake patented in 1869 by George Westinghouse, an American manufacturer. This system, in a slightly improved form, is still used on trains. Each railroad car has its own reservoir, called an auxiliary reservoir, connected by means of a valve with a brake pipe extending the length of the train. To apply the brakes, the engineer lowers the pressure in the brake pipe by letting air escape. The valve arrangement closes the connection between the reservoir and the brake pipe while simultaneously opening a normally closed connection between the reservoir and the brake cylinder. This allows the compressed air in the reservoir to enter the brake cylinder, forcing the piston in the brake cylinder against the train wheel, braking the train. To release the brakes, the engineer builds up the pressure in the brake pipe. This shifts the valve back to its former position and allows the air in the brake cylinder to escape, releasing the brake. An important advantage of this system is that any sudden drop in brake-pipe air pressure, such as that caused by cars uncoupling, will automatically apply the brake. The brake will continue to act until the problem is corrected, a good example of a fail-safe mechanism.

Another brake system with a fail-safe feature is used in heavy trucks. The brake is applied by means of a spring, and the power system is used to release the brake -- either by compressed air (an air brake) or by a vacuum (a vacuum brake). If the system should fail, the restraining force is removed, allowing the spring to apply the brake immediately.

A.5.5 Electric Braking

A machine powered by an electric motor may be designed with electric braking. The circuitry of the motor can be switched so that the motor operates as a generator driven by the rotating axle. Not only does the conversion of the rotational energy into electricity slow down the machine, but the electricity produced can be returned to the power source or collected in storage batteries, thus saving much energy. This general principle is called dynamic braking; the special case in which the electricity is returned to the source is known as regenerative braking.

A.6 POWER TRAIN

The power train includes a transmission, manual or automatic; a *clutch*, on cars with manual transmission; a drive shaft; a differential; and wheel axles. Although most conventional American cars use a front-mounted engine to drive the rear wheels, an increasing number of newer models use front-wheel drive, where the weight of the engine on the front wheels provides better traction on slippery roads. In some cars designed particularly for bad weather and rough roads, the engine drives all four wheels.

A.6.1 Transmission

The purpose of the transmission is to permit a change in the gear ratio between the engine crankshaft and the driven wheels. When the vehicle is started from rest, the ratio must be high so that the engine can turn at higher speeds and develop enough power to accelerate the car. As the vehicle speed increases, the transmission is shifted upward once or several times to decrease the gear ratio. The shift is accomplished in manual transmissions by a shift lever operated by the driver. The automatic transmission makes the shifts without driver intervention.

On cars with manual transmissions, a clutch -- a device for connecting and disengaging the engine -- is used to relieve the driving pressures through the transmission as gears are shifted. Internal clutches in the automatic transmission perform this function automatically.

A.6.2 Manual Transmission

In some smaller cars with small engines, four- or five-speed transmissions are used to compensate for the lower torque available from the engine. Trucks designed to haul heavy loads may have as many as 20 forward speeds and four speeds in reverse. The part of the transmission that houses the gears is called the gearbox. A manual transmission has a clutch to disconnect the engine crankshaft from the gearbox while shifting gears. The driver shifts gears by manipulating a shift lever, which is connected to the transmission by a mechanical linkage. There is, therefore, some choice in the location of the shift lever, which may be on the steering column or on the floor.

The older, sliding-gear transmission has largely been replaced by synchromesh transmission, in which synchronizers allow the gear teeth to be in constant mesh, turning freely on their shafts. The selected combination of gears is first synchronized (the teeth on the two gears are brought to the same speed of rotation) and then locked together so that power is transmitted to the drive shaft and then to the differential.

A.6.3 Automatic Transmission

Automatic transmissions use a torque converter--to couple the engine and the gearbox. It is a form of fluid coupling in which one rotating member causes the transmission fluid to rotate; the fluid, in turn, imparts a rotating motion to another rotating member on another shaft that is connected to the gearbox. The coupling of the torque converter is flexible, allowing slippage, for example, when the car is starting from rest and the wheels are not moving. As the car gains speed, slippage is reduced, and at cruising speeds the driven member turns almost as fast as the driving member. The gearbox contains a set of planetary gears, with clutches and brake bands for engaging the desired gears.

A.6.4 Drive Shaft

The drive shaft, or propeller shaft, connects the transmission with the differential, an arrangement of gears that allows the wheels to rotate at different rates when a car is turning. The drive shaft contains two types of joints: a slip joint and one or more universal joints. This allows the shaft to change its length and direction as the car wheels move up and down. In rear-drive cars with engines at the front, one drive shaft is needed. In front-drive cars with front-mounted engines, two are needed, one to each front wheel.

A.6.5 Wheel-Drive

Many vehicles now have full- or part-time four-wheel-drive (4WD). Part-time 4WD cars are driven in 2WD on paved roads. Most modern 4WD cars add an extra differential between the front and rear wheels, so the front and rear driving wheels can turn at minutely differing rates when driven on pavement, to avoid drive train damage.

In some vehicles with full-time 4WD, limited-slip differentials couple the front and rear final drive gears. Each driving axle has its own differential as well. These differentials allow front and rear wheels to turn at slightly varying rates to compensate for minor differences in tire diameters or drive-gear ratios. This slippage allows the vehicle with full-time 4WD to run on paved roads without damage to its drive train.

A.7 OTHER DESIGN CONSIDERATION

A.7.1 Catalytic Converter

A catalytic converter is a device in the exhaust system of an automotive engine that converts environmentally harmful exhaust gases into harmless gases by promoting a

chemical reaction between a catalyst and the pollutants. Catalysts are substances that speed or slow a chemical reaction between other substances, without themselves being consumed. Depending on the type of catalyst (oxidation, reduction, or dual), the catalytic converter decreases the emission of hydrocarbons and carbon monoxide, of nitrogen oxide, or of all three. A dual-catalyst system consists of an oxidation catalyst (for hydrocarbons and carbon monoxide) and a reduction catalyst (for the oxides of nitrogen).

In the most common type of catalytic converter, the exhaust gases are passed through a bed, or honeycomb, of small beads coated with the catalysts platinum and palladium. When the converter is heated and extra air is pumped into it by an air pump, contact with the catalysts causes the hydrocarbons and the carbon monoxide to be converted into harmless carbon dioxide and water. The reduction catalyst in the nitrogen oxide converter reduces the nitrogen oxides by splitting the nitrogen from the oxygen so that nitrogen gas, carbon dioxide, and water are formed. In an automobile equipped with a catalytic converter, lead-free gasoline must be used in order to prevent coating the catalyst with lead.

A.7.2 Muffler

The muffler's main function is to reduce engine noise to an acceptable level. Engine noise is a jumbled collection of its fundamental firing frequencies, which range from about 100 to 400 hertz (1 Hz = 1 cycle/sec); overtones of these; and an extended range of "white noise" caused by resonance of the various components.

A muffler attenuates noise in three ways. Interior compartments called Helmholtz tuning chambers are tuned to cancel resonance of specific frequencies. Others, called broadband dissipaters, are designed to reduce the energy of sound pulses and thus to attenuate a wide range of frequencies. Finally, the muffler's absorptive surfaces function like sound-deadening wall and ceiling panels to absorb noise. In a typical "three-pass" design, the exhaust stream changes direction twice as it passes through separate compartments, each tuned to attenuate certain frequencies. By the time the exhaust gases finally pass through the exhaust system (tail pipe); their temperature, pressure, and noise have been greatly reduced.

A.7.3 Aerodynamic Design

Designers of high-speed cars must also take into account other aerodynamic concepts such as the boundary layer. This is the layer of air nearest the skin of the car where the effects of the turbulence caused by air resistance are exhibited most strongly. It is desirable to keep turbulence to a minimum, so cars are designed to keep the stream of air flowing around the car as undisturbed as possible--hence the term streamlining.

A.7.4 Electric Car

An electric car is an automobile powered by an electric motor that is run by batteries.

The concept is not new. The essential battery technology was developed in the late 19th century, and many such cars were being manufactured by 1900. Although some models achieved high speeds for that time, the electric car was generally relatively slow, heavy, and expensive to operate. Its range was also limited by its dependence on facilities to recharge the battery. The use of the electric car diminished in the second decade of the 20th century with the invention of the self-starter, which made it unnecessary to hand-crank automobiles powered by gasoline.

Interest in the electric car revived after World War II with the development of smaller electric motors, more efficient batteries, and the advent of the space age. Lighter materials also became available for the chassis of such automobiles. However, car manufacturers were and still are reluctant to develop the electric car because of the many technical problems. For example, the distance an electric car can travel before its batteries must be recharged is limited despite advances in battery technology that have increased this range. The maximum cruising speed is also limited, as is the number of accessories that can be placed in the car. On the other hand, the electric car is mechanically more dependable and durable than the gasoline-powered car. It also does not pollute the atmosphere.

In-car battery recharge systems have not proved effective. An attempt to dispense with batteries altogether; and to design an electric-powered car without transmission, gears, or a drive shaft; shows greater promise. In this "hybrid" system, the main source of power is a gasoline-driven generator that powers small 18-kg (40-lb.) electric motors mounted behind each wheel. These "drive motors" accelerate the car and during deceleration generate more energy; which is stored in a flywheel mechanism, and used for heavy energy demands during acceleration.

A.7.5 Automotive Instrumentation

Automotive instrumentation performs the crucial role of monitoring vehicle operation and supplying information to both the driver and the vehicle subsystems. Mounted on the dashboard, standard instrumentation usually includes a speedometer that registers road speed, often coupled with an odometer to record accumulated distance and a tachometer showing the engine speed. Fuel and temperature gauges as well as oil-pressure and battery-charge warning lights complete the basic dashboard array. In recent years, however, the increased use of microcomputers to monitor engine performance and other automobile functions has led to far more sophisticated dashboard instrumentation. Some cars are now equipped with message centers that check basic mechanical conditions (for instance, engine and coolant temperatures, or the levels of transmission and brake fluid), and spell out a problem when it arises. To increase fuel economy, a monitor lights up when a change in gears will save gasoline. Other devices, linked to a computer, can announce instantaneous gas mileage, or tell the driver whether there is enough fuel to cover a certain distance.

Car makers anticipate even more elaborate instrumentation in the near future when the standard car will utilize several microprocessors, which may be capable of monitoring everything from tire air pressures to the alertness of the driver.

A.7.6 Safety

Automotive safety is concerned with reducing the number of traffic accidents and lessening the severity of injuries when accidents do occur. Areas of safety activity include the design of roads and highways, adjustments in laws pertaining to traffic and vehicles, systems of traffic control, programs of driver education, and vehicle design. Vehicle design gradually improved throughout the history of the automobile industry; higher speeds and heavier traffic, nevertheless, added to a climbing accident rate. In an attempt to deal with the problem, the U. S. Congress passed (1966) a law that permitted the federal government to issue mandatory safety standards for cars, trucks, motorcycles, and other vehicles. Since that time, more than 50 safety standards have been imposed, regulating safety windshields, safety belts, head restraints, brakes, tires, lighting, door strength, and roof strength.

A seat belt is a strap -- usually a shoulder harness -- that restrains an occupant in the seat, preventing him or her from being thrown out of the seat during a sudden stop or change in direction. Fewer than 20 percent of automobile occupants routinely use safety belts, even though convincing evidence exists as to their value, and legislation has been passed that requires seat belts in all cars.

Because of the poor response of the driving public to devices that require their active participation, safety researchers have developed automatic, or passive, restraint systems, which protect occupants without any action on their part. Two basic types of passive restraints have been produced: (1) the automatic belt, fastens around the occupant when the car door is closed; and (2) the air bag - in a crash, two air bags -- one in the steering column and one in the right side of the dashboard -- pop out and instantly inflate, forming cushions that prevent the occupants from striking hard surfaces, such as the dashboard or windshield.

It is estimated that about 12,000 lives could be saved and tens of thousands of severe injuries prevented each year if all cars had automatic restraint systems. The U.S. Department of Transportation (DOT) proposed in 1977 that all new cars be equipped with such systems by model year 1984, although major emphasis was on airbags rather than automatic seat belts. Automobile manufacturers objected, principally because the cost of air bags is high: \$300 to \$1,000 per car, depending on the volume of cars outfitted. New air bag designs have reduced that cost considerably, however, and driver's side airbags have been available on many of the most expensive cars since the mid-1980s. Several companies (e.g. Chrysler Corporation) plan to equip all of their models with airbags, and other auto makers are expected to follow suit.

A DOT regulation (1984) required that all new cars be equipped with passive restraint systems by mid-1989. The requirement could be rescinded if states representing two-thirds of the U. S. population enact mandatory seat-belt-usage laws.

INDEX

- a posteriori preference probability, 373, 374
- abductive inference, 68, 71
- abstraction levels, 11, 246, 252
- active filter, 525
- adaptation, 353
- adaptive learning for successful design, 365, 499, 659, 667
- adiabatic process, 270
- administrative decision making, 24
- algebraic expressions, *see* arithmetic expressions
- algebraic structure, 7, 110
- algorithmic methods, 14
- alphabet, 247
- alternative solution paths, 164
- analysis, 9, 23, 144, 149, 160
- analysis mapping, 114, 161
 - continuous, 168
- analysis of variance (ANOVA), 371
- analysis stage, 37
- analysis transformation, 190, 191, 202
- AND/OR
 - tree representation, 294, 298, 551
 - tree search, 293, 653
- antecedent, 78, 492
- apparel design, 416, 424, 436, 669
 - bodice pattern, 436
 - bodice sloper, *see* body measurements
 - body measurements, 436
 - CAD packages, 436
 - patternmaking, 436
- arc length, 424, 425
- arc welding design, 58
- arithmetic expressions, closed-form, 423, 425
- artifact
 - coupled, 161
 - ideal, 161
 - redundant, 161
 - uncoupled, 160
- artifact catalogue, 154
- artifact complexity, 242
- artifact description, 190, 143
- artifact part, 190, 201, *see also* design part
- artifact space, 7, 12, 111, 114
 - basis for, 658
 - regular, 125
 - strictly regular, 126
- aspiration level, 220
- assembling
 - and the Pareto distribution, 285, 660
 - and the Zipf distribution, 286
- assembly
 - defect rates, 261
 - efficiency, 262
 - efficiency measure, 263
 - errors, 262
 - operation time, 262
- assembly time, 14, 660
 - measure, 255
 - statistical analysis of, 279
 - total, 255, 283
- assignments, 112, 225
- asymptotic equipartition property (AEP), 228, 230
- attribute operation
 - topological, 157
- attribute space catalogue
 - closure in, 158
 - topological closure in, 158
- attribute space, 7, 111, 143, 153
 - basis for, 12, 146
 - closure, 157
 - preferences in, 166
 - topological, 157
- attributes, 144, 152, 396, 463
 - basic, 152
 - functional, 12
 - qualitative, 493
 - structural, 12, 110, 147, 149
- automata theory, 215
- automated design and manufacturing system (ADMS), 499
 - parameters, 504
 - performance measures, 503
 - physical configuration, 500
- automobile design, 551, 632
 - body, 638
 - brake system, 642
 - engines, 632
 - functional attributes, 552

- power train, 644
- production rules, 553
- steering system, 639
- structural attributes, 551
- suspension system, 640
- synthesis using the design search algorithm, 562
- axiomatic design theory, 66, 651
- axioms, 200
- backward chaining, 30
- balloon model, 268
- Barkan and Hinckley method, 285, 660
- basic synthesis problem (BSP), 217, 218, 658
 - intractability of, 222
- Bayes' theorem, 373
- behavioral view, 26
- Bezier curve, 432, 416, 424, 445, 448, 669
- bicycles, 31
- bifurcation, 408
- Boltzmann's constant, 271
- bondgraphs, 114
- Boolean expression, 208
- Boolean gates, 90
 - AND, 90
 - EXCLUSIVE OR, 90
- Boothroyd, 256, 262, 266
- Boothroyd-Dewhurst DFA method, 282, 660
- bounded post correspondence problem, 180
- bounded rationality, 5, 20, 24, 187
- bounded rationality postulate, 13
- brain-storming technique, 46
- branch and bound, 57
- bridge design, 39
- CAD/CAM/CAE, 28
- CADAT, 14, 106, 300, 551, 662
- CAE packages, 33
- cantilever beam, 387, 409
 - constraint model of, 419
- car design, 313
- car horn, 149
- CARRY, 92
- Case-1, 14, 300, 662
 - analyzer, 304, 662
 - attachments, 310
 - builder, 305, 662
 - case, 302, 662
 - domain, 303
 - question, 304
 - reasoning engine, 304
 - session, 303
- case-based reasoning, 652
- case study approach, 189
- catalogue, 146, 651
- catalogue selection problem, 361
 - genetic algorithm, 362
- category theory, 7
- causal history, 491
- causality, 59
- checklist method, 46
- chips, 328, 330
- Chrimes, 31
- circuit partitioning decision problem, 332
- circuit partitioning optimization problem, 332
- circuit partitioning problem, 324, 354
 - branch and bound algorithm, 336
 - computational complexity, 332
 - computational results, 339
 - genetic algorithm, 356
 - grouping heuristic, 334
 - mathematical formulation, 330
- closed form equations, 78, 491
- closed sets, 158
- Closed World (CW) condition, 48
- closure and stabilization, 76
- closure operation, 155
 - topological, 155
- clustering, 321
- COAST algorithm, 176, 387, 405, 424, 430, 440, 445, 448, 457, 513, 522, 523, 529, 668, 669
 - methodology, 397
 - outline, 407
 - sensitivity analysis algorithm, 406
- common range, 276, 655
- composite curve, 424
- composite objects, 423
- computational complexity theory, 7, 171
- computer aided design (CAD), 232, 387, 423, 445
 - intelligent, 661, 670
- computer aided engineering (CAE), 45
- computer classroom design, 578
 - functional attributes, 580
 - production rules, 584
 - structural attributes, 578
 - synthesis using the design search algorithm, 601
- computer simulation, 366, 368, 499
- concurrent design, 660
 - blackboard, 661
 - communication, 661
 - control mechanism, 661

- intelligent architecture, 661
- modular design, 661
- concurrent engineering, 5, 11, 43, 63
- concurrent product development, 666, 670
- conjecture and refutation (C-R), 70
- conjunctive normal form, 208
- consequent, 78, 492
- consistency
 - in curve fairing, 428
 - in variational design, 388
 - strong, 402
 - sufficient condition, 403
 - transitive theorem, 404
 - uniqueness theorem, 404
 - weak, 402
- consistent design, 394, 434
- consistent solutions, 389, 400, 440, 668
- constrained basic synthesis problem (CBSP), 218, 225, 230
- constrained nonlinear optimization, 423
- constraint management, 388
- constraint satisfaction, 388
- constraint-based design, 388, 423, 425
 - of faired parametric curves, 416, 668, 669
- constraints, 11, 77, 143, 396, 464, 652
 - closed-form, 463, 493
- constraints on parametric curves, 423, 447, 452
 - arc length, 428
 - distance from another object, 423, 425, 426, 432
 - in apparel design, 438
- constructivist approach, 7
- continuation methods, 390
- continuity, 7, 145
 - at a functional property, 167
 - at a structural property, 168
- continuity problem, 171
- continuous analysis, 145, 387, 667
- continuous mapping, 143
 - between functional and physical domains, 656
- continuous synthesis, 145, 387, 667
- continuous trajectories, 399
- convergence, 145, 159, 392, 393, 406, 415, 669
 - of function decomposition stage, 165
- corn in a pipe problem, 51
- correct solution, 400
- creative solution, 48
- crisis, 70
- curvature, 425
- curve-crawl factor, 392
- curve fairing, 424, 447, 455
 - via the stiffness criterion, 439
- DANSER algorithm, 448
- Darwinian natural selection, 71
- data tag, 85
- decompose and recombine, 44
- decomposition, 9, 293, 294, 653
 - of functions, 163
- decomposition solution composition, 658
- deduction theorem, 201
- degree of falsifiability, 73
- derailing detection device, 52
- descriptive complexity measure, 174
- design
 - as scientific discovery, 6
 - conceptual, 43
 - artifact representation, 4
 - as problem solving, 6
 - as purposeful activity, 23
 - as scientific problem solving, 67
 - as social process, 61
 - axioms, 146
 - computational analysis of, 13
 - conceptual, 7
 - coupled, 654, 655, 656
 - creative, 32
 - descriptive properties of, 9
 - diagonalized nature of, 33, 394, 667
 - form of, 25
 - innovative, 32
 - method, 39
 - routine, 32
 - sequential and iterative nature of, 27, 388, 667
 - tentative, 20
 - ubiquity of, 22
 - uncoupled, 323, 655
 - well-structured, 7
- design complexity, 243
 - functional, 14, 241, 244, 667
 - structural, 14, 241, 244
- design configuration, 396, 524
- design consistency, 387, 394, 424, 425, 427, 447, 524, 656, 667, *see also* consistency
- definitions, 400
- examples, 513
- principle of, 143, 176, 394, 430, 656, 667, 668

- theorems, 403
- design description, 26, 190, 197, 200, 493
 - consistent, 198
 - semantics, 200
- design evolution
 - ontogenetic, 29, 188
 - phylogenetic, 29, 31, 188
- design facts, 188, 655
- design for assembly (DFA), 256, 279
- design for manufacture (DFM), 43, 67, 279, 655
- design for testability, 347
- design form, 245, 247
 - size of, 249
- design guidelines
 - algorithmic, 662
 - general purpose, 651
- design heuristics, 227, 659
- design iteration, 394
- design method, 6, 65
- design methodologies, 4
- design paradigms, 6, 9, 38
 - algorithmic design paradigm, 57
 - analysis-synthesis-evaluation design paradigm, 40, 163
 - artificial intelligence design paradigm, 58
 - case-based design paradigm, 44
 - cognitive design paradigm, 45
 - creative design paradigm, 46
- design parameters, 365, 368, 374, 464, 491
 - controllable, 504
 - qualitative, 504
 - quantitative, 504
 - uncontrollable, 504
- design part, 492, *see also* artifact part
- design problems
 - ill-structured, 24
 - impreciseness of, 24
 - intractability of, 25
 - satisficing nature of, 25
 - well-structured, 23
- design process
 - as a mapping, 143, 653
 - case studies, 551
 - categories, 32
 - completeness of, 13, 657
 - computational complexity of, 199, 207
 - computational models, 4
 - correctness of, 199, 205
 - cycle, 20, 463
 - decidability of, 199
 - evolutionary model of, 397, 463, 491, 551, 652, 653, 661
 - evolutionary nature of, 7, 13, 20, 29, 143, 176, 187, 353, 387, 396, 652, 654
 - knowledge-level, 188
 - non-monotonic nature of, 79, 493
 - quantitative analysis of, 254
 - scientific community metaphor, 19, 666
 - soundness of, 13
 - type-0, 189, 200
 - type-1, 189, 203
 - type-2, 189, 197, 203, 298, 493
- design process (DP) problem, 207
- design process complexity, 243
- design process decision problem, 208
- design process execution, 192, 202, 204, 397, 494, 498
- design process history, 78, 192, 653
- design process optimization problem, 208
- design process states, 190, 197, 198, 201, 204, 493, 655
 - failed, 191, 198, 203, 205
 - initial, 190, 197, 198
 - successful, 191, 198, 203, 205
 - terminal, 190, 191, 197, 198, 203, 204
- design process step, 191, 198
- design range, 273, 368, 655
- design requirements, 396, 397, 524
 - categories of, 23
 - empirical, 23
 - extraction of, 24
 - functional, 365, 499, 652, 655
 - functionality, 23
 - ill-defined, 24
 - modifiability, 23
 - performance, 23
 - qualitative, 491
 - reliability, 23
 - well-defined, 23
- design search algorithm, 298, 551
- design search problem, 166, 652
- design solutions, 154
- design space, 7, 111
 - operators, 11
- design specifications, 20, 143, 153, 394, 463, 491
 - qualitative, 78, 493
 - quantitative, 493
 - tentative, 20
- design stage
 - conceptual, 652

- detailed, 37, 652
- design style, 57
 - bottom-up, 61
 - in apparel design, 436
 - meet-in-the-middle, 61
 - top-down, 61
- design theory, 6, 65
 - consistent, 6
 - descriptive, 5, 6
 - domain-independence, 6
 - empirical, 5
 - general, 6
 - mathematical, 7
 - prescriptive, 5, 6
 - simple, 6
- designing effort, 246
- deterministic design selection, 659
- differential gear, 31
- dimensions of a part, 388, 396
- distance between two entities, 165
- drip cleaner, 256
- dynamic programming, 57
- ECOBWEB, 323
- effort complexity measure, 253
- electric light bulb, 116
- electrical connections, 328, *see also*
 - microelectronics circuits
- electrical engineering, 85, 524
- electrical receptacle, 260
- electronic computers, 31
- elementary mental discriminations, 272
- embodiment, 43
- Empirical Programme of Relativism (EPOR), 75
- empirical research, 8
- Engineering Design Research Center, 64
- engineering design
 - logic of, 19
 - methodology, 9, 19
 - models, 7
- engineering knowledge
 - domain-specific, 659
- entity set, 146
- entropy of a random variable, 225, 250, 271, 658
 - joint, 250
- Euclidean space, 143
- evaluation, 9
- experimental success probability, 373, 510
- explanatory model, 188
- extension strategy, 49
- factor interactions, 366
- factors, 365
- fairing objective function, 416
- fastener design, 614, 621
 - functional attributes, 615, 622
 - production rules, 615, 624
 - structural attributes, 614, 621
 - synthesis using the design search algorithm, 620
- feasibility limits, 392
- finite state automata, 197, 207
- first law of thermodynamics, 270
- first-order logic, 7
- fitness value, 362
- fix point, 165
- fixed configuration problems, 361
- fixed step continuation, 390, 415
- flexible manufacturing cell, 85
- flexible manufacturing cell design, 499
- flexible manufacturing system (FMS), 101, 274, 361, 365, 366, 499, 667
- flow time, 504
- forklift design, 564
 - functional attributes, 566
 - production rules, 568
 - structural attributes, 565
 - synthesis using the design search algorithm, 576
- Formal Design Theory (FDT), 3, 109, 143, 651, 670
- formal language, 7
- function levels, 163
- function sharing, 44
- function space, 12, 114, 143, 153
 - basis for, 12, 143, 146, 173, 658
 - closure, 155
 - preferences in, 166
 - proximity in, 154
 - topological, 155
- function space catalogue
 - closure in, 156
 - topological closure in, 156
- function space character, 173, 174
- function-structure-behavior, 59
- functional description, 105
 - close, 12, 144
- functional domain, 143
- functional properties, 150, 152, 651
- functional requirements, independence of, 654
- functional view, 26

- functions, 143, 147
 - basic, 152
 - decomposable, 146
- gates, 330
- Gauss-Seidel iteration, 406, 418
- Gauss-Seidel operator, 403, 419
- gear box design, 77, 97
- General Design Theory (GDT), 146
- General Problem Solver (GPS), 30
- generalized topology, 7
- genetic algorithms, 354, 371, 666
 - alleles, 362
 - crossover, 355
 - decoding, 354
 - encoding, 354
 - evaluating a solution, 354
 - genes, 362
 - local optimum, 354
 - mutation, 355
 - parent solutions, 355
 - population, 354
 - reproduction, 355
 - termination, 355
- geometric modeling, 232, 389
- geometrical CAD systems, 389
- geometry, 387
- global optimization, 57
- global optimum, 416, 448
- goal directed problem solving, 30
- Gordon technique, 46
- gradient of the distance function, 426
- grand problem solving, 60
- graph, 124
- graph isomorphism, 180
- graphical objects, 423
 - bounded, 425
 - unbounded, 425
- group technology, 321, 328, 665
 - mix-and-match, 666
 - modular design, 666
- guided combinatorial analysis, 218
- guided heuristics, 217, 235
- Hanson, 71
- helical compression spring design, 521, 539
- hierarchical causal representation, 652
- hierarchical knowledge structure, 146
- Hitachi DFA method, 280
- hoist, 97, 361, 464
- holism, 7
- homeomorphic spaces, 7, 169
- homeomorphism, 145, 169
- homeomorphism problem, 172
- homotopy, 387, 434, 440, 523, 668
 - convex, 389, 400, 401, 416
 - function, 400
 - variable, 400
- horizontal shared memory, 63
- hypothetico-deductive (H-D), 68
- I/O pins, 328
- idea provoking techniques, 46
 - division, 50
 - increasing variability, 51
 - multiplication, 50
 - unification, 50
- idealized design process, 11, 143, 162
- I-DEAS Master Series, 33
- immature science, 69
- incompleteness postulate, 13
- inconsistency, 402
- incrementalism, 25
- independence axiom, 66
- infeasible regions, 409
- inference rules, 20
- information axiom, 67
- information content, 246, 249
 - functional, 273, 654, 655
 - minimal, 251
 - structural, 250, 275, 654, 655
- information retrieval systems, 143
- information theory, 7
- initial solution, user-defined, 392
- input-output models, 113
- integrated circuits (IC), 325, 665, *see also*
 - microelectronics circuits
- integration of physical parts, 654
- intelligent advisory tool, 293
- intended design solution, 394, 429, 445
- interactive curve design, 424
- interactive design system, 429
- interference, 261
- interpretative flexibility, 76
- inter-stage design iteration, 33, 668
- interval analysis, 403, 417
- interval analysis techniques, 400, 405
- interval continuation methods, 387, 389,
 - 416, 668
- interval equations, 418
- interval extension
 - of the Jacobian, 406
 - of the mean-value theorem, 405
- interval step control method, 406

- interval vectors, 418
- intra-stage design iteration, 37, 668
- iterative constraint solver, 390
- Kepler, 8
- knowledge acquisition, 61
- knowledge body, 188, 191, 464, 491, 653
- knowledge representation
 - entity-relational, 11, 109
 - extensional, 175, 651
 - intensional, 175, 651
 - nested-hierarchical, 11, 109
- knowledge-based CAD, 4
- knowledge-based expert systems (KBES), 59, 661
- Kuhn, 5, 20, 38, 65, 68
- Kuhnian paradigm, 38
 - disciplinary matrix, 38
 - exemplars, 38
- Lakatos, 20, 68
- languages, 179
- large scale integration (LSI), 346
- lasts, *see* models of a foot
- Laudan, 20
- learning synthesis knowledge, 322
- line tangent to two circles, 533
- local optimum, 416, 426, 448
- logic, 7
- logic design
 - eastern school, 39
 - western school, 39
- logical inference, 155
- Lucas DFA method, 281
- manual selection, 393
- manufacturing information, 145
- mass customization, 445
- Mathematica, 393, 414, 415
- mathematical programming, 57
- mature science, 5
- means-ends analysis, 60
- mechanical fasteners, 93, 192, 147
- mega-gates, 334
- metric, 145, 395
- microelectronics circuits, 345
 - logical design, 324, 346
 - packaging, 325
 - partitioning, 324, 325, 346, 665, 666
 - physical design, 324, 346, 666
 - placement, 325, 346
 - reliability, 347
 - routability, 347
 - routing, 324, 325, 346
 - schematic, 324
 - wiring congestion, 347
 - wiring length, 347
- microprogrammed control, 220
- Model 'T', 31
- models, 144
- models of a foot
 - 3-D virtual, 445, 447, 669
 - 3-D, 445
 - creation of virtual, 448
- modules, 220, 658
 - atomic, 112
 - basic, 11, 110
 - complex, 11, 110, 112
 - legal, 114
 - order, 126
 - primitive, 225
 - types of, 113
- modus ponens, 200
- morphological method, 46
- morphology, 394
- motor drive assembly, 289
- Motorola, 263
- multi-objective decision criteria, 367, 370
- multi-objective design problems, 367
- multiple competing solutions, 387, 388, 394, 430, 513
- natural science, 23
- natural selection, 353
- n-dim project, 64
- nearly decomposable system, 74, 665
- necessary conditions of nonlinear programming, 424, 427, 429, 433
- needs assessment, 9
- neighborhood, 172, 395
 - of a functional property, 159
- neighborhood system, 172
- net, 328, 330
- net terminals, 328
- Newell, 30, 71
- Newton, 8
- Newton-Raphson method, 388, 392, 414, 424
- non-determinism postulate, 13
- nonlinear algebraic equations, 388
- nonlinear programming optimization, 362, 448
- norm...l science, 69
- NP-complete problem, 25, 171, 208, 222, 332
- NP-hard problem, 209

- object oriented methods, 110
- objective function, 362, 426
- open sets, 158
- operands, 110, 246, 247
- operators, 110, 246, 247
 - closure composition, 129
 - closure decomposition, 132
 - closure integration, 133
 - composition, 11, 110, 126
 - decomposition, 11, 110, 130
 - evaluation, 219
 - first-order closure composition, 129
 - first-order closure decomposition, 131
 - indicator, 128
 - integration, 11, 110
 - k_1 - k_2 closure integration, 133
 - k_1 - k_2 order integration, 133
 - projection, 128
 - proper composition, 128
 - restriction of a first-order composition, 131
- optimization search techniques, 362
- order preserving
 - in attribute space, 169
 - in function space, 169
 - under synthesis, 169
- order relation, 165
- orthogonal array, 371, 376, 506
- output performance measures, *see* responses
- over-constraining the system, 393

- parallelism between science and design, 75
- parameter levels, 365, 368, 374, 667
- parametric design, 388, 463
- parametric objects
 - points, 423
 - arcs, 423
 - curves, 423
 - lines, 423
- performance metrics, 165
- philosophy of science, 9, 19
- physical domain, 143
- piston subassembly, 120
- P-learning algorithm, 366, 374, 499, 667
 - application to flexible manufacturing cell design, 506
 - catalogue structure, 381
- pneumatic piston subassembly, 263
- point at a distance from two points, 528
- pooling strategies, 377

- Popper, 20, 70, 189
- positivists, 70
- power, 271
- power law, 270
- predictor-corrector, 391, 405, 668
- preference probability, 373, 374, 510
- preference structure, 220
- preferences over sets, 167
- pressure, 268
- printed circuit board (PCB), 329, 346
- Pro/Engineer, 33
- probabilistic design selection, 227, 659, 667
- probabilistic paradigm, 365
- probability distribution of the output
 - performance measure, 369
- probability of success, 273, 369, 374, 499, 655
 - overall, 370, 667
- problem characteristic variables, 48
- production planning, 526
- production rules, 188, 190, 197, 202, 204, 293, 302, 492, 493, 551, 653
- programmable logic controller (PLC), 90
- proof, 201
- propositional calculus (-logic), 124, 143, 178
- protocol analysis, 45
- protocol studies, 30
- prototype model, 44
- psychology, 10
- purposeful adaptation, 29

- quadrature techniques, 424, 428
- Qualitative Change in Problem Characteristic (QC) condition, 48
- quality function deployment (QFD), 43

- rapid prototyping, 34
- rationality
 - principle of, 244
 - pure, 24
- real knowledge, 12
- redesign, 31
 - incremental, 31
 - innovative, 31
- reductionism, 7
- redundancy, 161
- refined upper bound, 227
- reflection-in-action, 68
- reinforced concrete building, 118, 219
- relations, 11, 110, 112, 220, 225, 658
 - unit, 126
- relaxation factor, 392, 415

- representative sample, 370, 374
- research programme, 69
- responses, 273, 365, 368
- restructuring strategy, 49
- result-oriented, 23
- revolution and resolution, 70
- robotic manipulator, 526
- robust design, 43
- routine solution, 48
- rule-based systems, 143, 362

- satisfactory performance, 5
- satisfiability problem (SAT), 238
- science of the artificial, 23
- scientific communities, 20
- scientific discovery, 68
- search
 - backward, 11
 - best-first, 166, 298, 653
 - blind, 298
 - breadth-first, 57, 166, 298, 653
 - depth-first, 57, 166, 298, 653
 - exhaustive, 57
 - forward, 11
 - greedy, 57
 - heuristic, 214, 298
 - rapid, 57
 - stochastic, 365
- second law of thermodynamics, 270
- serial binary adder unit, 92, 247
- serial optimization, 57
- set theory, 139
- shared meaning, 62, 63
- shared memory, 63
- shoe design, 416, 445, 668
 - best fit selection, 446
 - custom, 445
 - foot measurements, 445, 451
 - pattern, 445
 - style, 446
 - upper pattern grading and visualization, 447
- Simon, 4, 23, 24, 29, 61, 71
- simplicity, 242
- Simpson's quadrature formula, 439
- simulation language, SIMAN, 506
- small and medium scale of integration (SSI/MSI), 346
- smallest probable set, 366
- social constructivist approach, 62
- sociology of science, 75
- software complexity, 243
- software engineering, 242
 - object-oriented models, 526
- solid fuel rocket engine, 55
- solid model, 233
- solution space, 105, 293
- solution trajectories, 398, 401, 430, 441, 513, 668
- specification description, 191
- specification part, 191, 492
- specifications
 - initial, 188
 - presumed, 191, 201, 204
 - validated, 191, 201, 204
- state space, 166, 293
- statistical experimental design, 365, 499
- statistical mechanics, 271
- stepwise refinement, 30
- Stroud, 253
- Stroud number, 272
- structural description, 105
 - close, 145
- structural properties, 152, 651
- structural view, 26
- structure space, 7, 111
- Structured Inventive Thinking method (SIT), 46, 48
- Suh, 66, 242
- SUM, 92
- supercritical fluid chromatography, 96
- surface model, 233
- switching circuit, 115
- symbolic medium, 246, 247
- synthesis, 9, 23, 145, 149, 160
- synthesis mapping, 161
 - continuous, 167
- synthesis states, 397
- synthesis transformation, 190, 191, 202
- system neighborhood objects, 48
- system of constraints, 388, 394, 397, 416, 424, 430, 445
- system of equations, 390
 - fully constrained, 400
- system of nonlinear constraints, 387, 463
- system of nonlinear equations, 387, 424, 430, 447
- system range, 274, 655
- systemhood, 7
- systems science, 7

- Taguchi's statistical method, 512
- task/episode accumulation process, 45
- testing stage, 20

- theory-oriented, 23
- thermodynamic process, 268
- thermodynamics, 10
- thinghood, 7
- 3-CNF decision problem, 208
- 3-PARTITION decision problem, 332
- time complexity measure, 253, 283, 660
- tire design, 604
 - functional attributes, 607
 - production rules, 608
 - structural attributes, 606
 - synthesis using the design search
 - algorithm, 612
- tolerance limits, 153
- tolerances, 153, 273, 368
- tools of verification, 78
- topological property, 170
- topological spaces, 7, 176
 - closure, 146
- topological structure, 146
- topology, point-set, 146
- Torbeck valve, 31
- transformations, 197, 198, 204, 493, 494, 653
 - between descriptions, 23
 - between function and attribute spaces, 160
 - coupled, 654
 - uncoupled, 654
- TRIZ, 47
- tumor problem, 54
- Tycho de Brahe, 8
- typical set, 366

- uniform probability distribution, 274
- unijunction transistor metronome, 258
- unintended solution, 389
- universal upper bound, 225

- variational CAD systems, 400
- variational design, 77, 387, 416, 423, 424, 495, 667
 - evolution in, 396
- Ve Cone valve, 31
- verifications tools, 491
- vertical shared memory, 63
- very large scale integration, 346
- volume, 268

- weak methods, 30, 60
- wireframe feature recognition, 217, 233, 659
 - connectivity problem, 235, 659
 - feature recognition problem, 236, 660
- wireframe model, 232
- work, 269
- work in process (WIP), 86, 504
- wormgear assembly design, 514
 - design execution, 514
- wormgear reducer design, 97, 463, *see also*
 - gear box design
 - accessories, 487
 - bearing, 485
 - casing heat, 488
 - casing, 477, 487
 - concepts, 466
 - key, 484
 - load and strength constraints, 475
 - motor, 470
 - shaft, 479, 481
 - transmission parts, 471
 - wear-resistance, 486
 - constraint model of, 548
- Yoshikawa, 30, 146
- zero information state, 370

Applied Optimization

1. D.-Z. Du and D.F. Hsu (eds.): *Combinatorial Network Theory*. 1996
ISBN 0-7923-3777-8
2. M.J. Panik: *Linear Programming: Mathematics, Theory and Algorithms*. 1996
ISBN 0-7923-3782-4
3. R.B. Kearfott and V. Kreinovich (eds.): *Applications of Interval Computations*. 1996
ISBN 0-7923-3847-2
4. N. Hritonenko and Y. Yatsenko: *Modeling and Optimimization of the Lifetime of Technology*. 1996
ISBN 0-7923-4014-0
5. T. Terlaky (ed.): *Interior Point Methods of Mathematical Programming*. 1996
ISBN 0-7923-4201-1
6. B. Jansen: *Interior Point Techniques in Optimization. Complementarity, Sensitivity and Algorithms*. 1997
ISBN 0-7923-4430-8
7. A. Migdalas, P.M. Pardalos and S. Storøy (eds.): *Parallel Computing in Optimization*. 1997
ISBN 0-7923-4583-5
8. F.A. Lootsma: *Fuzzy Logic for Planning and Decision Making*. 1997
ISBN 0-7923-4681-5
9. J.A. dos Santos Gromicho: *Quasiconvex Optimization and Location Theory*. 1998
ISBN 0-7923-4694-7
10. V. Kreinovich, A. Lakeyev, J. Rohn and P. Kahl: *Computational Complexity and Feasibility of Data Processing and Interval Computations*. 1998
ISBN 0-7923-4865-6
11. J. Gil-Aluja: *The Interactive Management of Human Resources in Uncertainty*. 1998
ISBN 0-7923-4886-9
12. C. Zopounidis and A.I. Dimitras: *Multicriteria Decision Aid Methods for the Prediction of Business Failure*. 1998
ISBN 0-7923-4900-8
13. F. Giannessi, S. Komlósi and T. Rapcsák (eds.): *New Trends in Mathematical Programming. Homage to Steven Vajda*. 1998
ISBN 0-7923-5036-7
14. Ya-xiang Yuan (ed.): *Advances in Nonlinear Programming*. Proceedings of the '96 International Conference on Nonlinear Programming. 1998
ISBN 0-7923-5053-7
15. W.W. Hager and P.M. Pardalos: *Optimal Control. Theory, Algorithms, and Applications*. 1998
ISBN 0-7923-5067-7
16. Gang Yu (ed.): *Industrial Applications of Combinatorial Optimization*. 1998
ISBN 0-7923-5073-1
17. D. Braha and O. Maimon (eds.): *A Mathematical Theory of Design: Foundations, Algorithms and Applications*. 1998
ISBN 0-7923-5079-0

CHAPTER 21

CONCLUDING REFLECTIONS

This book is a treatise on the design process and the formal approach to design. The basic premise of the formal approach to design is that there are basic principles that govern decision making in design, just as the laws of nature govern the physics of artifacts.

The work is made up of four parts: (Part I) a largely philosophical discussion of engineering design; (Part II) Formal Design Theory (FDT), which deals with algebraic representation of design artifacts, a model of the idealized design process, a model of the evolutionary design process, a discussion of complexity and probabilistic search approach, general measurable metrics for evaluating a good design, and a study of Boothroyd-Dewhurst's design efficiency and total assembly time. This theory is leading to several useful outcomes; (Part III) a methodological validation by means of algorithms and heuristics for design decision support has been developed to show that the proposed theory is consistent and fruitful; (Part IV) powerful and comprehensive case studies have been provided to show how the various formulations are linked to genuine design problems, their underlying knowledge bases, and their concomitant information processing.

The aim of this chapter is to recapitulate the *principal* corollaries or facts that are direct consequences of FDT's axioms and theorems (Section 21.1), and the derived algorithms and case studies (Section 21.2). These corollaries tend to resemble the flavor of design *guidelines*. The following guidelines either validate common design rules and are found to be consistent with other design theories (e.g., the axiomatic theory of design, see [1]), or provide new insight and design criteria.

21.1 GENERAL PURPOSE GUIDELINES

21.1.1 REPRESENTATION OF DESIGN KNOWLEDGE

GUIDELINE 1 (*Extensional* versus *Intensional* Representation). There are two potential representations for expressing design knowledge as presented in Chapters 4 and 5: *extensional* and *intensional*. In the extensional representation approach, structural or functional properties are expressed by means of a set of artifacts in a *catalogue*, which have these properties. In the intensional representation approach, artifacts are described by a complete set of structural or functional properties,

including their relations, which together describe an artifact that delivers the functional requirements and satisfies the constraints.

FDT proposes the use of both representations in design support systems. Extensional descriptions are particularly useful in case-based reasoning. In case-based reasoning, designs are obtained by searching a memory bank of previous cases for a design that solves a similar problem. When implementing a computationally efficient case-based reasoning system, extensional descriptions work best when the most similar design to the current problem is chosen. Thus, indexing cases with a rich vocabulary of the structural properties and design functions they satisfy is a key to effective case-based reasoning. When designing engineering artifacts, the design cases need to be indexed in terms of the output behavior of interest. Matching and retrieval can be driven by associative processes on these indexes.

Extensional descriptions enable the recognition of similarity among almost identical products. They can also be easily modified to include new cases and properties. Thus, extensional descriptions can support newfangled ideas although they cannot generate them themselves. Nevertheless, in real design it is recognized that perfect mappings of the design process, from design specifications to completed devices or components, do not exist. However, FDT postulates that human designers use hierarchical causal representation to relate the structure of the device to its function by means of causal if-then structured networks. Thus, employing intensional descriptions may result in better support of such knowledge structures. Intensional representations do not support the similarity between products, but have more concise descriptions and are fundamental to supporting incremental (evolutionary) design.

The ability of extensional descriptions to explore a memory bank of previous artifacts that satisfy similar specifications is important in the conceptual or preliminary design stages. In the detailed design phase, refinements that involve assigning attributes to parts and sizing parts may be better dealt with intensionally. This involves creating the intensional representations of entities, which would link the attributes to the parts, the functions to the attributes, and the assignment of values to these attributes. After detailing the intensional descriptions, the process terminates with a feasible solution.

21.1.2 DESIGN PROCESS

GUIDELINE 2 (Evolutionary Task Structure). The design problem is formally a search problem in a large space for objects that satisfy multiple requirements and constraints. Only a small number of objects in this space constitute satisficing or optimal solutions. What is needed for practical design are strategies that radically shrink the search space.

In Chapter 6, a systematic top-level method -- the *evolutionary design model* -- that guides the search in the solution space was presented. This approach involves a set of functional requirements and constraints (both constitute the design specifications). Some specifications serve as the input to the design process as a

whole; the others are generated as sub-specifications. Given a design specification, a set of knowledge and control strategies in the form of *facts* and *production rules* are postulated, and a structured set of *transformation operators* are suggested as a means of achieving a specification. Each transformation operator involves, through the activation of associated links, the retrieval of facts and production rules from the knowledge body, and the application of certain of these rules. The latter results in the *inference* of either a partial design solution that satisfies the specification; or the addition of new facts, rules, or sub-specifications. Eventually, a complete solution will emerge. The set of transformation operators leading to this final, complete solution constitutes the *history of a design process*.

The characteristics of the evolutionary design model may establish the framework within which different information mapping processes are examined (see Part III). These mapping processes can be examined based on different design domain boundaries (e.g., customer to functional, functional to physical, physical to process planning): (1) a mapping series of mapping that converts a set of customer requirements (needs) to a vector of design functional requirements and constraints (specifications in the functional domain); (2) starting with the stated functional requirements and constrain, the designer generates a solution by defining design attributes (design parameters in the physical domain) that satisfy the functional requirements and constraints. This synthesis process constitutes a mapping between the functional and physical domains; (3) the design attributes generated in the physical domain are interpreted as a set of requirements for implementation in the manufacturing process domain. The design attributes are achieved by a set of process attributes.

GUIDELINE 3 (Use of Effective Control Strategies). The solution path from initial specifications to the physical artifact is not unique. There can be innumerable plausible solution paths, yet a selection needs to be made. For instance, if abstract specifications decompose so that enough information is provided to proceed to the subsequent design stages (see Chapter 10), then two sets of control issues need to be considered. The first control issue deals with which sets of decompositions to choose. A decomposition will generally produce an AND node. All the sub-specifications in an AND node will need to be solved, but only one of the decompositions will need to be solved (an OR node). Finding the appropriate decomposition requires a search in an AND/OR graph. Three representative methods of guiding this search are: (1) breadth-first, (2) depth-first, and (3) best-first. Best-first methods state that decompositions are driven by a form of *hill climbing*, where there is a particular decomposition (an AND node) in the direction of maximal increment based on some measure of overall performance. The second control issue involves the order in which the sub-specifications within a given decomposition ought to be attacked. The main constraint is the knowledge about dependencies between sub-specifications. If there are independent specifications, one must satisfy the specifications in order of importance (from most to least).

GUIDELINE 4 (Accomplish Functions Prior to Constraints). The design problem is

specified by a set of functions to be delivered by an artifact and a set of constraints to be satisfied. A computationally effective process of design is to generate a candidate design based on functions, and then to modify it to meet the constraints.

GUIDELINE 5 (Maintain the Independence of Functional Requirements). As stated in Guideline 3, in an evolutionary design process there can be innumerable plausible solution paths (set of transformation operators). However, not all proposed solution paths are good solutions. The criteria for differentiating between a “good” transformation (from a current design state to a modified design state) and a “bad” transformation is by capturing creative knowledge. *Uncoupling* means that each transformation exhibits a minimum number of overlapping functional requirements. The following uncoupling criterion is a direct consequence of FDT: the aim of a good transformation strategy is to uncouple the current functional requirements so that each modified functional requirement or partial design solution (e.g., device, component, or design parameter) affects only one set of functional requirements. If the independence of functional requirements is violated (coupled transformation), the probability of achieving a design solution that satisfies the initial specifications decreases. Thus, coupled transformations can never have minimal functional information content (see Chapter 8). In addition, coupled transformations may lead to complex dependencies among specifications. This might involve making commitments to partial design solutions (subparts) and later revising them when other functional requirements and constraints (for other subparts) are violated. Thus, coupled transformations may lead to a design process that is not computationally effective (deviating from Guideline 2).

GUIDELINE 6 (De-coupling of Coupled Design, see also [1]). When the number of attributes (design parameters) in an artifact is less than the number of functional requirements, either a coupled design results, or the functional requirements cannot be satisfied. The design may be de-coupled by the addition of new design attributes to make the number of design attributes equal to the number of functional requirements.

GUIDELINE 7 (Minimization of Structural Information Content). In Chapter 8 we defined a structural design complexity measure as a function of the *information content* associated with the design’s representation. The design’s representation may include facts, causal relations, mathematical models, etc. Thus, the following principle results in a computationally effective design process: among all proposed transformations from a current design state to a modified design state that satisfy the uncoupling criterion (as stated in Guideline 5), the optimal transformation has the lowest structural information content.

GUIDELINE 8 (Minimization of Functional Requirements and Integration of Physical Parts). The following principles can be inferred from the minimum structural information principle stated in Guideline 7, we infer the following principles: (1) a good transformation strategy (from a current design state) minimizes

the number of functional requirements and constraints in the modified design state. This implies that given two process states; $\langle M_1, \theta_1 \rangle$ and $\langle M_2, \theta_2 \rangle$, where $\langle M_1, \theta_1 \rangle$ embodies design facts and functional requirements in addition to those embodied in $\langle M_2, \theta_2 \rangle$, the structural information content associated with $\langle M_1, \theta_1 \rangle$ is less than that associated with $\langle M_2, \theta_2 \rangle$. In other words, as the number of functional requirements and constraints increases, the system becomes more complex and thus raises the structural information content. This implies that the design process will be more difficult to operate and maintain (computationally) than one that only meets the stated functional requirements at each transformation state; (2) At the end of a design process all specifications have been implemented, and the initial specifications (functional requirements and constraints) have been converted into a physical artifact. Thus, an artifact should fulfill the precise needs defined by the specifications -- not more and not less. This implies that the conventional cliché, "My design is better than yours because it does more than was intended," is misguided. Reliability may also decrease, due to increased complexity when an artifact fulfills more functional requirements than are required; (3) the number of physical parts and the relations among them should be reduced in order to decrease the structural information content. This should be done when possible without coupling functional requirements (thus violating Guideline 5). In other words, the structural information content of uncoupled design solutions with fewer components is less than those with more components that satisfy the same set of functional requirements and constraints.

GUIDELINE 9 (Minimization of Functional Information Content, see also [1]). In Chapter 8, we defined functional information as the specification of what a symbol structure (e.g., an artifact or design process) should be able to do. That is, information has a purpose, and this enables the design to attain goals. Defining information content in functional terms means that the capabilities of each solution alternative may be compared with the governing set of requirements until the designer identifies the solution alternative that best satisfies the functional requirements. Following this definition, we defined the *functional information content* of an artifact as a function of the probability of its successfully achieving the functional requirements (abbreviated as the *probability of success p*). Specifically, functional information content is defined as the logarithm of the inverse of the probability of success p . Thus, the following principle provides a rational means for selecting the best design: among all proposed solutions that satisfy the uncoupling principle (Guideline 5), the optimal design has the lowest functional information content. This implies that, depending on whether the solution is created in the physical or process domain, the optimal solution will have the highest probability of successfully achieving the functional requirements or design parameters.

GUIDELINE 10 (Decreasing the "System Range", see also [1]). In Chapter 8, we introduced a measure of functional information content, which is defined in terms of the "design range," "system range," and "common range". The probability of

satisfying the functional requirements is expressed in terms of the amount of overlap (“common range”) between the “design range” and the system capability (“system range”). To reduce the functional information content (following Guideline 9), the designer has to either increase the “common range” or decrease the “system range”.

GUIDELINE 11 (Design for Manufacturability, see also [1]). In order to reduce the functional information content of a product or process (following Guideline 10), the following design rules should be utilized (depending on whether the design is created in the physical domain or manufacturing process domain): (1) use standardized or interchangeable parts; (2) use symmetrical shapes and/or arrangements; (3) eliminate adjustments, fasteners, jigs and fixtures; (4) specify the largest allowable tolerance when stating functional requirements. Thus, given two or more uncoupled designs that satisfy the same set of functional requirements and constraints, the design with the largest tolerance (“design range”) is superior. In all cases, these rules should be utilized if they are consistent with the functional requirements and constraints. The basic idea behind these design rules is to establish set of “common ranges” within which the product or manufacturing process is capable of meeting functional requirements (expressed in terms of tolerances).

GUIDELINE 12 (Information Content of Coupled versus Uncoupled Designs, see also [1]). The uncoupling principle stated in Guideline 5 is related to the minimum functional information principle as follows: in coupled designs, functional requirements can be satisfied by affecting other functional requirements. The interactions among functional requirements tend to amplify the “system range,” and thus result in an increase in the functional information content. Therefore, it is desirable to maintain the independence of functional requirements.

GUIDELINE 13 (Functional Information Content of Uncoupled Design). If each functional requirement is independent of other functional requirements, the functional information of the artifact is the sum of the functional information content of all the individual functional requirements that must be satisfied.

GUIDELINE 14 (Principle of Design Consistency). In an ideal design, small changes in specifications should lead to small changes in design and vice versa. Therefore, if the current artifact’s function differs slightly from the required function, a small modification to the structure may be sufficient. The mathematical concept used in Chapter 5 to investigate the principle of design consistency by *continuous* mapping between the *functional* and *physical* domains applies here.

GUIDELINE 15 (Design Consistency, Coupling, and Functional Information Content). The following guideline states that coupled designs, when the principle of consistency is satisfied, result in an increase of the functional information content. Assume that the principle of design consistency is satisfied, and the state of design parameters is slightly changed from one state, $D^0 = (d_1^0, d_2^0, \dots, d_N^0) \in \mathfrak{R}^N$, to

another state, $D^1 = (d_1^1, d_2^1, \dots, d_N^1) \in \mathfrak{R}^N$, in the physical domain. The principle of design consistency implies that the system behavior is such that the changes in the functional requirements, $F^0 = (f_1, f_2, \dots, f_N) \in \mathfrak{R}^N$, due to a change in design parameters may be written as follows (see Figure 21.1). For each functional requirement f_i ,

$$\Delta f_i = \frac{\partial f_i}{\partial d_1}(d_1^1 - d_1^0) + \frac{\partial f_i}{\partial d_2}(d_2^1 - d_2^0) + \dots + \frac{\partial f_i}{\partial d_i}(d_i^1 - d_i^0) + \dots + \frac{\partial f_i}{\partial d_N}(d_N^1 - d_N^0) \tag{21.1}$$

In uncoupled design, the change in the design parameter d_i is such that $\frac{\partial f_i}{\partial d_i}(d_i^1 - d_i^0)$ is the desired change in f_i . In coupled design, the deviation from this ideal condition (constituting the “system range”) is given by the *other* terms on the right hand side of the equation. Therefore, in coupled designs, where functional requirements can be satisfied by affecting other functional requirements, the interactions among functional requirements tend to amplify the “system range.” When the designer-specified tolerances for functional requirements are given, uncoupling results in an increase in the functional information content. Therefore, as stated in Guideline 12, it is desirable to maintain the independence of functional requirements.

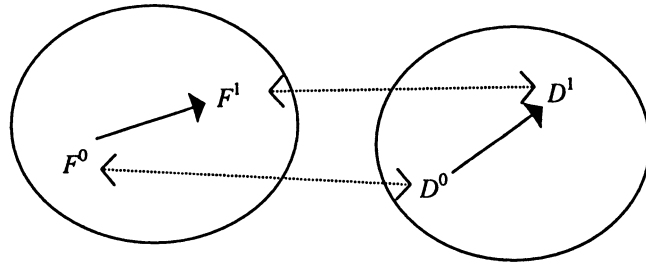


Figure 21.1 Small Changes in Specifications Lead to Small Changes in Design and Vice Versa

GUIDELINE 16 (Completeness of the Design Process). As discussed in Chapters 6 and 10, the design process involves mapping the initial functional requirements through a series of transformations that lead to a physical description. The series of transformations exhibits precedence among its transformed states. A functional requirement may correspond to a partial design solution or decompose into sub-requirements. A set of general production rules is used (1) to decompose requirements into sub-requirements; or (2) to match the requirements with components. The Completeness guideline states that the set of production rules, which are associated with the decomposition of requirements into sub-requirements

and/or components, must be exhaustive over the domain of interest. That is, recreating a new set of production rules to accommodate a new design situation or scenario should not be needed. There are many alternate sets of production rules that can be utilized for the life-cycle domain, but some may be redundant. Most life-cycle situations can be managed with a “complete basic set” of production rules. In Chapter 6, we demonstrated that the computational complexity of establishing a “complete basic set” of production rules is NP-Complete, which means that there exists no polynomial time algorithm capable of solving the problem. Thus, the potential to solve the problem depends of the availability of certain heuristics.

GUIDELINE 17 (Decomposition-Solution Composition). Consider a scenario where a “complete basic set” of transformations was found as suggested in Guideline 16. Assume that it is known that the designer might encounter many design problems (each described by a set of functional requirements) during the life-cycle domain. In such a scenario, prior to carrying out the design process (on-line), it would be an efficient strategy to perform off-line preprocessing that involves finding a “complete basic set” of functional requirements. When the designer encounters a new design problem, the given set of functional requirements needs to be translated into a set that is included in the “complete basic set” of functional requirements. If sub-problems are solved, the sub-problem solutions are “glued” into the solution to the original design problem. This approach suggests the use of past cases to make new design decisions rather than solving each new problem from scratch. The decomposition-solution composition idea is comparable to the concept of utilizing a *basis* for the artifact and function spaces as introduced in Chapter 5.

GUIDELINE 18 (Expressiveness versus Complexity). In Chapter 7, a basic synthesis problem was specified by (1) a set of functional requirements to be delivered by the artifact; and (2) a repertoire of components (modules) assumed to be available and a vocabulary of relations between components. The solution to the basic synthesis problem involves specifying a complete set of components and their relations. Together these aspects need to describe an artifact that delivers the functional requirements.

A solution to the basic synthesis problem is characterized by three parameters: (1) the possible number of relations that each component can share, which grows polynomially (of order γ) with respect to the number of basic components N ; (2) the number of relations of interest, ρ ; and (3) the maximum number of occurrences, v , of any relation. It was shown that the total number of solutions having these characteristics is bounded by,

$$|\mathcal{S}| \leq v \rho^2 \rho N^{\gamma+1} H_0\left(\frac{v}{N^{\gamma+1}}\right) \quad (21.2)$$

where $H_0(P)$ denotes the binary entropy function $H_0(P) = -p \log_2 p - (1-p) \log_2 (1-p)$.

Thus, we conclude that although expressiveness of the design problem (partially

determined by the above parameters) is necessary for the generation of a wide variety of solutions, increasing the expressiveness of a design problem may swamp the designer with alternatives. So, any increase in expressiveness needs to be accompanied by an increase in the designer's ability to control the complexity of the design space.

GUIDELINE 19 (Decomposition and Complexity Reduction). As shown in Chapter 7, the following engineering design heuristics, based on the specialized circumstances relevant to artifacts, will enable the replacement of the formidable universal upper bound presented in (21.2) with a much smaller upper bound:

- Heuristic1.* the number of relations of interest is bounded by some $\rho \ll N$;
- Heuristic2.* the maximum number of occurrences of any relation satisfies $v \ll N$.

If there is effective decomposition of a given design problem into smaller sub-problems, and each satisfies the foregoing heuristics, then the combinatorial complexity of the original problem can be significantly reduced.

GUIDELINE 20 (Uncertainty and Complexity). As mentioned in Chapter 7, the nature of the information involved in the search for a design solution may be *deterministic*, by showing which design decisions are categorically inferior; or *probabilistic*, by identifying those design decisions with the greatest probability of solving the problem. In the latter case, the designer's problem may be not knowing which decisions will ultimately lead to a satisfying design solution. In this case, the appealing proposition we established in Chapter 7 comes to light. Not all possible solutions (i.e., the accumulation of a series of design decisions) have the same probability of solving the problem; and the probability of identifying a successful solution is inversely proportional to the number of design decisions, and to the uncertainty associated with each decision. Thus, *learning* about which design decisions satisfy the governing functional requirements reduces the uncertainty associated with the decision, which in turn increases the probability of identifying a successful solution to the problem.

GUIDELINE 21 (Complexity of Wireframe Feature Recognition). In Chapter 7, we stated that in order to recognize features (i.e., a collection of geometric entities) from wireframe models it is necessary to: (1) preprocess the database to derive topological connectivity; and (2) determine which connected subsets correspond to which features of interest.

The following engineering design heuristics, which are based upon domain-specific engineering knowledge, provide combinatorial upper bounds that indicate that the connectivity algorithm upon large databases will be computationally tractable. For N geometric entities:

- Heuristic1.* the number of geometric entities per feature $\ll N$;
- Heuristic2.* the number of features of interest $\ll N$;

The following additional design heuristic, which is based upon domain-specific engineering knowledge, presents a quadratic upper bound for the number of connected subsets that may be sent to the feature recognition module:

Heuristic3. the features of interest are disjoint, and are sparsely distributed over the part.

GUIDELINE 22 (Time Complexity Measure). In Chapter 9, we inspect through extensive statistical analysis the correlation between the time complexity measure T (see Chapter 8) and the estimates of product assembly times that were derived by Boothroyd and Dewhurst in their Design for Assembly (DFA) structured methodology [2]. The correlation between the time complexity measure T and Boothroyd and Dewhurst's estimates is found to be very close to +1 over a wide diversity of experiments. This demonstrates that the time complexity measure T may be used as a powerful predictive tool. Such a tool could be used in the earliest stages of concept development to estimate total assembly times, or to compare competing concepts. Thus redesign may be stimulated while it is easiest to make design changes.

GUIDELINE 23 (Minimize and Simplify Assembly Operations). The time complexity measure T reveals two fundamental factors that can contribute to assembly time: (1) the number of assembly operations (a subset of assembly interfaces), which affects the time complexity measure more than linearly; and (2) the number of parts (a subset of assembly operations). Thus, part count alone is not adequate for defining design simplicity or predicting conformance quality. Instead, a superior design criterion is: to minimize and simplify assembly operations. This tends to reduce part counts and simplify part interfaces while avoiding assembly complexity that may be introduced to achieve part count reduction. Since one source of assembly defects is the interference between mating parts, this strategy must also be pursued to reduce the assembly defects in quality products.

GUIDELINE 24 (Assembling and the Pareto Distribution). The time complexity measure is found to be consistent with the hypothesis by Barkan and Hinckley [3] that the set of manual assembly times per operation follows a Pareto distribution. That is, the shortest assembly times per operation are the most likely to occur. Using this distribution of assembly times, it has been shown by Barkan and Hinckley that the total manual assembly time of a product can be bounded as a function of the part count and operation count.

GUIDELINE 25 (Concurrent Design). Concurrent design is different from that of serial design. The serial product design assumes a one-way progression along linear paths, with no procedural feedback or iteration and no looping or back-chaining. In concurrent design, the information flows bidirectionally to upstream and downstream tasks. To illustrate, if the accurate information is not made available to the product design team by the customer, how can the product be supported in the field? Similarly, if a design team does not know the limitations of the available

manufacturing processes, how can the team design the product in a cost-effective manner? The same can be said for the downstream tasks. How can a manufacturing group produce a part if the manufacturing process is not accurately defined or the design is not well-understood? Thus, a concurrent design requires that the design team consider process constraints while the process planners consider design constraints.

We identify two factors that should be considered to incorporate concurrency into a design process:

1. "Modular Design" - a product is designed in a "modular" fashion if the work of development is partitioned among designers (or design teams) who work independently. Following the uncoupling criterion presented in Guideline 5, the design tasks should be partitioned by rearranging the assignment of tasks to reduce interdependence across functional groups. However, several obstacles can prevent this ideal from occurring. One obstacle can arise from the sharing of results; for example, one team uses results from another team as its input. Another obstacle can occur when two tasks need each other's results as input. These "input dependent" problems are solved through the use of assumptions. Each task uses a "best guess" at what the other task's result will be and proceeds. As intermediate results are obtained, they are substituted for the assumption. When an assumption proves to be false or falls outside an agreed-upon range, the other task is notified and new assumptions are negotiated. This method allows both tasks to proceed.
2. Communication - another method of concurrent design that minimizes the cost of coordinating activities across design teams is by facilitating inter-functional communication. To accomplish the necessary communication between tasks and resolve possible conflicts, a synchronizing and mediating activity is required at certain process points or after specific periods of time. This resolution process makes the status of each task explicit, and provides alignment of the dependent tasks. It is an active problem-solving activity. In order to pursue specialization economies by choosing more design groups, firms need to develop better communication technology that enables to reduce the coordination loss across design teams.

To achieve the goals outlined above, we propose the use of an Intelligent Concurrent Design architecture. The proposed concurrent architecture can be envisioned as a network of users supported by Intelligent CAD systems, where the communication and coordination is achieved through a global database (Blackboard) and a control mechanism. The Blackboard is the medium through which all communication takes place. Each Intelligent CAD system can be viewed as a knowledge based expert system developed for solving individual design tasks, and supported by standard CAD tools (e.g., a database structure, an analysis program). Each Intelligent CAD operates according to the evolutionary design process model as delineated in Chapters 6 and 10, as well as Guideline 2. The Intelligent Concurrent

Design architecture may also be implemented within design groups.

21.2 ALGORITHMIC DESIGN GUIDELINES

Based on the theory and the guidelines derived above, algorithms and heuristic methods have been developed. Methods include *case based reasoning*, *group technology*, *genetic algorithms*, *adaptive learning*, *probabilistic search for successful design*, and *continuation methods for maintaining design consistency*. It is the aim of this section to summarize these methods, and delineate how the above guidelines were used as a pre-processor for the systematic methods presented in Part III.

21.2.1 LOGIC DECOMPOSITION AND CASE BASED REASONING (CHAPTER 10)

Our investigation of the validity of the design process model as introduced in Chapter 6 has been conducted within a case-based reasoning framework for design (called CAse-based Design Advisory Tool, see Figure 21.2), and has been implemented in the Case-1™ system. Case-1 has a memory that contains a set of general production rules, also termed as *cases*. The cases are derived from previously designed artifacts, and are used to guide the design process. Requirements are decomposed into sub-requirements and/or the corresponding structural properties. Cases can be retrieved from the memory using several indexing techniques to make searching the knowledge base more efficient and effective.

Case-1 involves up to three steps that allow for user interaction: case builder, case analyzer, and evaluation. The first two steps that are currently implemented are summarized as follows:

- *Case Builder* is the authoring tool used by knowledge engineers to create and modify cases. *Cases* are the atoms of the Case-1 system. They may be thought of as a single potential decomposition of a requirement into sub-requirements or corresponding structural attributes. A case consists primarily of a *problem description* part (a requirement) and a *solution* part (potential sub-requirements or structural attributes). Some examples are provided in Table 21.1 based on a problem of fastener choices.

™ Case-1 is a trademark of ASTEA International, 1995.

Table 21.1 Knowledge Representation in Case-1

problem description	“strength rivet high & holding rivet variable & length rivet variable & adjustability rivet permanent & insertion-case rivet easy”
solution	“socket-hex drive”
problem description	“strength high”
solution	“strength-material high & head-size large & thread-size large & threaded length large”

A case provides information that allows Case-1 to determine if the case is relevant to a problem the user might present. This information is comprised of the textual description of the problem and solution, a list of qualifying questions and answers, and any appropriate hypermedia attachments. Case-1 ranks the case based on the answers the designer enters. The results are then used by the Analyzer to provide the user the most probable set of cases that apply to the particular problem. A domain is a set of cases that pertain to a particular problem area (e.g., fasteners). The designer selects the domains to be searched during each session.

- The *Analyzer* performs case retrievals. To do this, the designer starts a new session and types the requirements in plain English. The Case-1 Reasoning Engine presents a list of possible cases through its knowledge of the existing production rules and several indexing techniques (that consider the relevance of the case and common words). This process is *interactive* and can be repeated several times to link requirements to the sub-requirements through the case hierarchy until it links them to the structural attributes of the product. After the session is complete, the session can be saved and re-opened later. By reviewing the unsatisfied specifications, the designer can then determine the physical components that satisfy the initial design specifications, and thus uncover a design solution alternative. Sessions that do not result in a satisfactory decomposition of requirements need to be reviewed by a knowledge engineer to ensure proper decomposition in future sessions.

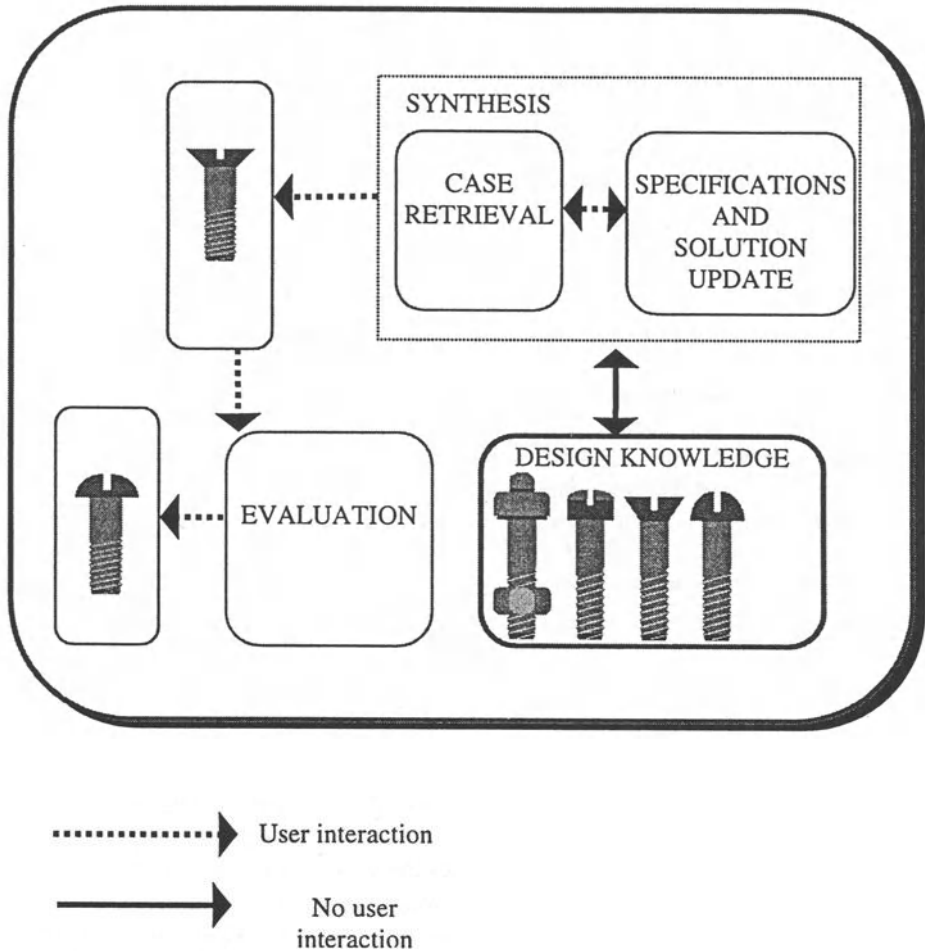


Figure 21.2 The CADAT Architecture

The proposed approach has the following characteristics:

- It provides a systematic approach that guides the search in a large and complex solution space.
- It uses a set of production rules, based on previous “good” designs, to guide the search.
- Each production rule has associated information decomposed and structured such that the relevance of each production rule to any requirement can be determined.
- New design knowledge can be incorporated into the existing knowledge base without modifying the existing production rules.

- The existing production rules can be changed relatively independently of other rules.
- Each production rule may have one or more hypermedia attachments that provide additional information to the designer.
- The scoring of production rules with respect to a particular requirement is dynamically adjusted by the system to reflect the actual operational experience.

21.2.2 GROUP TECHNOLOGY AND CLUSTER ANALYSIS (CHAPTER 11)

Group Technology is a planning concept in which similar products are grouped and then produced together in order to exploit their similarities. Generally the objective is to group products according to the constraints of the collective tool or processing requirements, and to organize them into as few groups as possible. The formation of part families leads to the reduction of design variety, and procedures that are rationalized and automated.

In Chapter 11 we suggested two major methods in which Group Technology may be used in design:

1. Diversity of products is the major cause of problems for designers. Each product satisfies its own functional properties, and requires its own design attributes, process plan, etc. It is possible that “similar” products are designed at different times by different design personnel. To avoid this, existing designs have to be retrieved and used as the prototypes for designing the new products. If this retrieval task is left to the designers, it is almost impossible to totally avoid impreciseness and excessive time to design. Thus, we introduced in Chapter 5 the notion of a “distance” between two entities in the function or attribute spaces. Defining a distance metric will enable the grouping of entities (artifacts, structures, or other entities) into classes that reflect the commonality of certain properties (e.g., structural attributes, functional attributes). When introducing a new design problem (specification properties), the synthesis system may try to incorporate it into the existing classification. At this stage synthesis should terminate and candidate designs should be generated.

2. In Guideline 5 we presented the uncoupling criterion for judging a “good” design from a “bad” design. A design that adheres to the uncoupling criterion may also be considered a nearly decomposable system. That is, a system of components C_1, \dots, C_n where each C_i is an aggregate of more primitive entities such that the interactions between the entities within the C_i 's are appreciably stronger than those between the C_i 's. For example, electrical circuit partitioning, also called packaging or assignment, involves transforming a drawing of the logical circuit into sub-circuits. Sub-circuits are assigned to, or packaged in, a module (also called *integrated circuits*). For a given partitioning, the number of interconnections between modules is used as a measure of uncoupling (an efficiency measure). The smaller the measure, the better the design. In Chapter 11, we examine the applicability of grouping and clustering techniques to the partitioning problem of electronic circuits in order to

achieve uncoupled (or decomposable) solutions.

Group Technology may also be applied to the study of “modular design,” and its effect on the performance of concurrent product development projects. A product is “modular” if development is partitioned among designers (or design teams) who work independently. The modules are connected by standardized interfaces that permit designers to “mix and match” modules that maximize measures of performance. Often finer product partitions improve productivity by specializing but increase the costs of coordinating the activities. Intuitively, we wish to group together any two components C_i and C_j that have high interaction between them. The higher the interaction between the two components, the higher the return to grouping C_i and C_j together. On the other hand, when there are fewer components to design, design teams can achieve larger gains by specializing. By selecting a design group partition and engaging in product design research, we attempt to maximize the expected overall performance of the product.

21.2.3 SOLVING DESIGN PROBLEMS WITH GENETIC ALGORITHMS (CHAPTER 12)

In Chapters 2 and 6, we presented the biological and scientific community’s evolutionary metaphor applied to design, which models the observed evolutionary phenomenon that occurs between the time when a problem is assigned to the designer and the time the design is passed on to the manufacturer. During this period, the design evolves and changes from the initial form to the acceptable form, and we say that there is a fit between the design and the requirements. An evolutionary metaphor takes into consideration concepts of randomness of generation, selection based on fitness for survival, populations of solutions, propagation of members of populations based on some evolutionary mechanism, representation based on genotypes and prototypes, etc.

The term “evolutionary transformation process” also refers to the generally accepted biological (genetic) evolutionary transformation process, where mechanisms such as crossover and mutation are used on entire populations. Based on this meaning, we demonstrated and formulated in Chapter 12 the applicability of genetic algorithms to the partitioning phase of electronic circuit design. The results suggest that a genetic algorithm can find the optimal solution in a large percentage of the problems, particularly for smaller data sets. The genetic algorithm results were significantly better than the sequential chip construction heuristic presented in Chapter 11. Both in terms of the average and maximum percentage deviations from the optimal value. The use of genetic algorithms in the context of the physical design of printed circuit boards seems to be a promising area for future research.

Configuring a design from parts out of catalogs, which satisfy predetermined goals and constraints, requires searching large numbers of part combinations. This makes it difficult and impractical to do an exhaustive search for the best solution using this technique. Rule-based approaches suffer from being domain specific and

thus are not appropriate as general solution methods. However, genetic search techniques seem to be an encouraging approach for solving fixed configuration problems as shown in Chapter 12.

21.2.4 PROBABILISTIC SELECTION METHODS FOR SYSTEM DESIGN (CHAPTER 13)

In Chapter 13, we addressed the problem of the parameter design of complex systems. Design solutions are characterized by their parameter settings (each fixed to one out of various possible levels). The design's functional requirements are represented by a set of pre-specified limits that determine where the output responses should fall. The designer has to identify those designs that maximize the likelihood of satisfying a given set of functional requirements. The proposed method is based on the *functional design complexity measure* provided in Chapter 8 for quantifying how well a proposed artifact satisfies the governing requirements (in probabilistic terms). We consider situations where the exact analytic relationships between the design parameters and the design responses are unknown or can not be practically determined. Consequently, the designer has to evaluate these relations by statistical inferences of experimentation results.

A methodology for adaptive learning of successful designs (termed as P-learning) is proposed. The P-learning algorithm constructs a sequence of samples (populations of candidate solutions), where each sample includes particular designs that it can simulate. Thus, the P-learning algorithm can learn more about the design's behavior through the samples. In particular, this helps determine which parameter-levels appear to satisfy the governing requirements in terms of overall success probability. As the desired information is obtained, the P-learning algorithm generates new candidate solutions (sample elements) with a bias toward candidate solutions that include better parameter levels. In Chapter 18, a real industrial problem of designing a flexible manufacturing system that is presented and solved based on the P-learning algorithm.

21.2.5 MAINTAINING CONSISTENCY IN THE DESIGN PROCESS (CHAPTERS 14, 15, 16)

In Chapter 5, we investigated the notion of design consistency (Guideline 14 above): *small changes in specifications should lead to small changes in design*. The mathematical concept that is used to investigate the principle of design consistency is that of continuous analysis and continuous synthesis. Our research and contributions (in Chapters 14, 15, and 16) further develop the areas of general design consistency as applied to variational design, incorporating curves in variational design, and shoe design CAD.

In Chapter 2, the sequential and iterative nature of design was introduced and the diagonalized design framework was presented as an alternative approach to

integrating the two disparate views of the design process. The diagonalized design framework represents the design process as constantly iterating among design stages. In an ideally flexible design environment, even when the design is nearly finished, the designer can still make modifications at the conceptual design level. Good design, should attempt to maximize the inter-stage iteration and minimize the intra-stage iteration. One means towards the latter goal is to allow more realistic design constraints in the design system. However, when more powerful constraints are allowed, there are often multiple ways of satisfying the constraints. In such an environment, the ability to maintain design consistency is of paramount importance. Often, maintaining design consistency is assumed by the user, and any design environment would be deemed unusable if it was not available.

In Chapters 14, 15, and 16, we reviewed design consistency in variational design, constraint-based curve design, and 3-D shoe design systems. Design consistency in any interactive design system is typically approached by requiring the user to identify the specific rules for deciding which solution (among many) is the correct solution. The problems with this approach are that it is typically very computationally intensive to find all the possible solutions, and it is often very hard or even impossible to characterize the solutions so as to arrive at the correct one regardless of how the specifications are modified. A different approach is to use continuation methods to track a given solution, rather than trying to characterize all the possible solutions. While our technique, COAST, is based on this idea, continuation methods have typically been used for following a curve with the “predictor-corrector” paradigm. It is unknown, with such methods, if the constraint solver converges to the correct solution or not. In the area of constraint-based curve design, we reviewed advanced and interactive techniques for creating or modifying a curve. There was no research found, however, that allowed constraint-based design of curves with constraints on the whole curve, rather than on defined points on the curve. Finally, we looked at shoe design systems and found a class of software available for creating 2-D upper pieces and a class of software for displaying 3-D surfaces; However, no software was found that performed both jobs (requiring the designer to input the design twice).

In Chapter 14, we implemented our COAST method for maintaining design consistency and demonstrated its application to several variational mechanical design examples. If there is only one possible solution to the specifications, then it is easy to maintain a consistent design. It is much harder when there are multiple competing solutions that all satisfy the specifications. Fortunately, Guideline 14 directs us towards a principle of design consistency: small changes in specifications should lead to small changes in design. Starting from an initial system of constraints, an initial satisfactory solution, and a change in constraints, COAST is able to track the desired solution as the constraints are changed. A homotopy is setup between the two systems of constraints, and interval continuation methods are used to deterministically follow the trajectory of the desired solution. Interval techniques allow COAST to guarantee that it will either converge to the consistent solution or detect those situations in which it cannot do so. COAST was applied to example designs of a worm gear assembly and a cantilever beam. In both cases, COAST was

able to converge to the consistent solution when the constraints were modified.

In Chapter 15, we illustrated with COAST the ability to perform constraint-based curve design. Rather than attempting to constrain all the individual points on the curve, we demonstrated how to constrain the behavior of the whole curve. Constraints between the desired curve and another object take the form of an optimization problem. In this case, we reduced the optimization to its set of necessary conditions. Constraints of the intrinsic behavior of a curve take the form of an integral expression over the length of the curve. In this case, we used quadrature techniques to reduce the integral to a nonlinear equation. Constraints were also reduced to nonlinear form. If there were fewer constraints than unknowns, we faired the curve using another constrained optimization reduced to its set of necessary conditions. We then setup a similar homotopy as before and extended the COAST method to handle modified coefficients of the constraints as well as modified right hand side of the constraints. These techniques were applied to curve design examples involving bezier and b-spline curves and to apparel design.

Finally, in Chapter 16, we demonstrated the capability of creating a consistent 3-D virtual last with COAST. First, a set of 52 measurements were derived that described a general last. Then, a series of 22 3-D Bezier curves were constrained to form the outline of the last such that the 52 measurements (as well as other constraints) were satisfied. Finally, when the measurements were changed, the technique was shown to be able to converge on a new solution that was consistent with the original solution.

We believe the major contributions of these methods are:

- The diagonalized design framework for managing the sequential and iterative nature of design.
- The general mathematical approach to maintaining design consistency in an evolutionary design system whenever the similarities of designs can be quantified.
- The specific algorithm (COAST) for maintaining design consistency when the dimensions of the design are constrained through a system of nonlinear equations. Given an initial system of constraints and an initial solution, COAST is able to follow the desired solution when the constraints are modified.
- The demonstration of the importance of maintaining a local optima in an interactive design system, and the application of the COAST method to the maintenance of a desired local optimum.
- The new approach to incorporating curves into variational design systems. It is shown how to place relational constraints on curves like any other graphical element.
- The novel approach to creating a consistent 3-D virtual last when given an arbitrary set of measurements taken from the user's foot.

21.3 SUMMARY

This book suggests broad avenues in approaching research challenges that will enable the vision of Formal Design Theory (FDT) to be achieved. Some additional issues that have yet to be explored are summarized below:

- In designing a product (or a process), what are the most productive paths through the specifications and artifact trees?
- How can physics (engineering knowledge in the broad sense) be most beneficially incorporated between the specifications and artifact trees?
- Can the formal definition of concurrent engineering activities be included in our approach to design process modeling?
- Would it be useful to provide a powerful computational framework within the general evolutionary process model (developed in Chapter 6), which studies creativity and in which the creative design process can be described and explained?
- Can statistical methods improve the probabilistic design selection method presented in Chapter 13?
- Can the transition from design to manufacturing be modeled by mathematical mapping?
- Can the belief that “small design changes can lead to significantly increased manufacturing cost” be captured by the discontinuity of mathematical mapping?
- How can FDT research benefit other design methodologies, such as Quality Function Deployment (QFD)?

Much work must still be done to reach the goal of developing Intelligent CAD systems that help at the conceptualization stage (incorporating manufacturing, materials and assembly information), as well as the implementation stage of developing detailed design attributes. The future development of FDT must also address large concurrent product development projects. A collaborative engineering project typically involves a group of designers working cooperatively on distributed tasks, and striving to complete a complex design by closely interacting among themselves and dynamically sharing design data and information.

It is hoped that this book presents FDT's concepts in a clear manner that will benefit designers and researchers through the insight FDT provides. In addition, perhaps the material will inspire readers to undertake the challenge of further developing the theory and applying it to yield more effective design processes and products.

REFERENCES

1. Suh, N.P., *The Principles of Design*. New York: Oxford University Press, 1990.
2. Boothroyd, G. and Dewhurst P., *Product Design for Assembly*. Wakefield, RI: Boothroyd & Dewhurst Inc, 1987.
3. Barkan, P., and Hinckley, C. M., “The Benefits and Limitations of Structured Design Methodologies,” *Manufacturing Review*, Vol. 8, No. 3, 1993.