Frontiers in Artificial Intelligence and Applications

MULTI-RELATIONAL DATA MINING

Arno J. Knobbe

IOS Press

MULTI-RELATIONAL DATA MINING

Frontiers in Artificial Intelligence and Applications

Volume 145

Published in the subseries

Dissertations in Artificial Intelligence

Under the Editorship of the ECCAI Dissertation Board
Proposing Board Member: Joost Kok

Recently published in this series

Vol. 144. P.E. Dunne and T.J.M. Bench-Capon (Eds.), Computational Models of Argument – Proceedings of COMMA 2006
Vol. 143. P. Ghodous et al. (Eds.), Leading the Web in Concurrent Engineering – Next Generation Concurrent Engineering
Vol. 142. L. Penserini et al. (Eds.), STAIRS 2006 – Proceedings of the Third Starting AI Researchers' Symposium
Vol. 141. G. Brewka et al. (Eds.), ECAI 2006 – 17th European Conference on Artificial Intelligence
Vol. 140. E. Tyugu and T. Yamaguchi (Eds.), Knowledge-Based Software Engineering – Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering
Vol. 139. A. Bundy and S. Wilson (Eds.), Rob Milne: A Tribute to a Pioneering AI Scientist, Entrepreneur and Mountaineer
Vol. 138. Y. Li et al. (Eds.), Advances in Intelligent IT – Active Media Technology 2006
Vol. 137. P. Hassanaly et al. (Eds.), Cooperative Systems Design – Seamless Integration of Artifacts and Conversations – Enhanced Concepts of Infrastructure for Communication
Vol. 136. Y. Kiyoki et al. (Eds.), Information Modelling and Knowledge Bases XVII
Vol. 135. H. Czap et al. (Eds.), Self-Organization and Autonomic Informatics (I)
Vol. 134. M.-F. Moens and P. Spyns (Eds.), Legal Knowledge and Information Systems – JURIX 2005: The Eighteenth Annual Conference
Vol. 133. C.-K. Looi et al. (Eds.), Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences – Sharing Good Practices of Research, Experimentation and Innovation

ISSN 0922-6389

Multi-Relational Data Mining

Arno J. Knobbe
Kiminkii, The Netherlands

IOS Press
Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

ISBN 1-58603-661-0
Library of Congress Control Number: 2006931539

Publisher
IOS Press
Nieuwe Hemweg 6B
1013 BG Amsterdam
Netherlands
fax: +31 20 687 0019
e-mail: order@iospress.nl

Distributor in the UK and Ireland
Gazelle Books Services Ltd.
White Cross Mills
Hightown
Lancaster LA1 4XS
United Kingdom
fax: +44 1524 63232
e-mail: sales@gazellebooks.co.uk

Distributor in the USA and Canada
IOS Press, Inc.
4502 Rachael Manor Drive
Fairfax, VA 22032
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

PRINTED IN THE NETHERLANDS

Multi-Relational Data Mining
A.J. Knobbe
IOS Press, 2006

Contents

This page intentionally left blank

Acknowledgements

As is customary for a Ph.D. thesis, the road towards completion of this text has been long. Two people have been instrumental in reaching the end successfully, and getting me started in the first place. I am very grateful to Pieter Adriaans for convincing me that my research ideas were a suitable basis for a dissertation, and that getting a degree was a mere formality and would be a matter of one or two years (a slight underestimate). Equal praise to my supervisor Arno Siebes, for supporting my ideas, having the patient conviction all would end well, and letting me do things my way. Whenever my research led me off the beaten track of mainstream Data Mining, he encouraged me to press on.

I would also like to thank my colleagues at the Large Distributed Databases (read 'Data Mining') group at Utrecht University, who, for obscure reasons, tended to come up with Spanish nicknames, ranging from *Arniño* to *Pensionado*. In particular Lennart Herlaar, Rainer Malik and Carsten Riggelsen were of great help in getting the document printer-ready. Ad Feelders, also at the LDD group, devoted his time reading through an early draft. I hope this was as beneficial to him as it was to me.

A greatly appreciated effort was done by Kathy Astrahantseff, who checked the manuscript for typos and bad phrasing. Thanks a lot for spending so much time crossing the t's and dotting the last ı's.

The person with probably the most visible impact on the book as you are currently holding it is Lieske Meima, who spent lots of here valuable spare time designing the cover and taking wonderful pictures.

Many of the experimental results in this thesis would have been impossible without the hard work of the team at Kiminkii: Eric Ho, Bart Marseille, Wouter Radder and Michel Schaake. I am particularly indebted to Eric and Bart for helping me implement Safarii and ProSafarii. Even though at times, they must have been wondering where all their efforts were leading, they can be proud of the end result.

Although at the end of the day, every letter in this thesis was conceived by me, a surprisingly small fraction of these letters was actually typed in person. Many thanks to Karin Klompmakers, Tiddo Evenhuis and Hans van Kampen for typing out endless pages of manuscript and sitting down with me to make corrections and draw tables and diagrams.
I want to express my gratitude to the members of the reading committee, Jean-François Boulicaut, Luc De Raedt, Peter Flach, Joost Kok and Hannu Toivonen, for voluntarily spoiling their summer carefully reading the manuscript and approving its publication.
Thanks also to my two assistants at the public defence of this thesis, Marc de Haas and Leendert van den Berg. Looking like a clown is best done in teams.
The following institutions have supported or contributed to the research reported in this thesis: Perot Systems Nederland B.V., the CWI (the Dutch national research laboratory for mathematics and computer science), the Telematica Institute, Utrecht University and Kiminkii.
Finally I have to mention my dad, Freerk Knobbe, who provided a lot of technical support and still recognizes randomly located paragraphs he claims to have typed. On many occasions, he helped out with tedious jobs such as creating an index or editing formulae in Word. His only complaint was that the randomness of his contributions prevented him from seeing the big picture and understanding the 'plot'. I guess with the present dissertation in print, he will have to read it start to finish.
But all the technical and scientific support would have been in vain if it hadn't been for the moral support provided by my friends and family, in particular my parents.

Houten, September 2004

Multi-Relational Data Mining
A.J. Knobbe
IOS Press, 2006

1 Introduction

This thesis is concerned with Data Mining: extracting useful insights from large and detailed collections of data. With the increased possibilities in modern society for companies and institutions to gather data cheaply and efficiently, this subject has become of increasing importance. This interest has inspired a rapidly maturing research field with developments both on a theoretical, as well as on a practical level with the availability of a range of commercial tools. Unfortunately, the widespread application of this technology has been limited by an important assumption in mainstream Data Mining approaches. This assumption – all data resides, or can be made to reside, in a single table – prevents the use of these Data Mining tools in certain important domains, or requires considerable massaging and altering of the data as a pre-processing step. This limitation has spawned a relatively recent interest in richer Data Mining paradigms that do allow structured data as opposed to the traditional flat representation.

Over the last decade, we have seen the emergence of Data Mining techniques that cater to the analysis of structured data. These techniques are typically upgrades from well-known and accepted Data Mining techniques for tabular data, and focus on dealing with the richer representational setting. Within these techniques, which we will collectively refer to as Structured Data Mining techniques, we can identify a number of paradigms or 'traditions', each of which is inspired by an existing and well-known choice for representing and manipulating structured data. For example, Graph Mining deals with data stored as graphs, whereas Inductive Logic Programming builds on techniques from the logic programming field. This thesis specifically focuses on a tradition that revolves around relational database theory: Multi-Relational Data Mining (MRDM).

Building on relational database theory is an obvious choice, as most data-intensive applications of industrial scale employ a relational database for storage and retrieval. But apart from this pragmatic motivation, there are more substantial reasons for having a relational database view on Structured Data Mining. Relational database theory has a long and rich history of ideas and developments concerning the efficient storage and processing of structured data, which should be exploited in successful Multi-Relational Data Mining technology. Concepts such as data modelling and database

normalisation may help to properly approach an MRDM project, and guide the effective and efficient search for interesting knowledge in the data. Recent developments in dealing with extremely large databases and managing query-intensive analytical processing will aid the application of MRDM in larger and more complex domains.
To a degree, many concepts from relational database theory have their counterparts in other traditions that have inspired other Structured Data Mining paradigms. As such, MRDM has elements that are variations of those in approaches that may have a longer history. Nevertheless, we will show that the clear choice for a relational starting point, which has been the inspiration behind many ideas in this thesis, is a fruitful one, and has produced solutions that have been overlooked in 'competing' approaches.

1.1 Data Mining

The primary ingredient of any Data Mining [20, 33, 43, 110, 112] exercise is the database. A *database* is an organised and typically large collection of detailed facts concerning some domain in the outside world. The aim of Data Mining is to examine this database for regularities that may lead to a better understanding of the domain described by the database. In Data Mining we generally assume that the database consists of a collection of *individuals*. Depending on the domain, individuals can be anything from customers of a bank to molecular compounds or books in a library. For each individual, the database gives us detailed information concerning the different characteristics of the individual, such as the name and address of a customer of a bank, or the accounts owned.
When considering the descriptive information, we can select subsets of individuals on the basis of this information. For example we could identify the set of customers younger than 18. Such intensionally defined collections of individuals are referred to as *subgroups*. While considering different subgroups, we may notice that certain subgroups have characteristics that set them apart from other subgroups. For instance, the subgroup 'age under 18' may have a negative balance on average. The discovery of such a subgroup will lead us to believe that there is a *dependency* between age and balance of a customer. Therefore, a methodical survey of potentially interesting subgroups will lead to the discovery of dependencies in the database. Clearly, a good definition of the nature of the dependency (e.g. deviating average balance) is essential to guide the search for interesting subgroups. Such a statistical definition is known as an *interestingness measure* or *score function.*
Interesting subgroups are a powerful and common component of Data Mining, as they provide the interface between the actual data in the database and the higher-level dependencies describing the data. Some Data Mining algorithms are dedicated to the discovery of such interesting subgroups. However, interesting subgroups are a limited means of capturing knowledge about the database, because by definition they only describe parts of the database. Most algorithms will therefore regard interesting subgroups not as the end product, but as mere building blocks for comprehensive descriptions of the existing regularities. The structures that are the aim of such algorithms are known as *models*, and the actual process of considering subgroups and laboriously constructing a complete picture of the data is therefore often referred to as *modelling.*
We can think of the database as a collection of raw measurements concerning a particular domain. Each individual serves as an example of the rules that govern this domain. The model that is induced from the raw data is a concise representation of the workings of the domain, ignoring the details of individuals. Having a model allows us to reason about the domain, for example to find causes for diseases in genetic databases of patients. More importantly, Data Mining is often applied in order to derive predictive models. If we assume that the database under consideration is but a sample of a larger or growing population of individuals, we can use the induced model to predict

the behaviour of new individuals. Consider, for example, a sample of customers of a bank and how they responded to a certain offer. We can build a model describing how the response depends on different characteristics of the customers, with the aim of predicting how other customers will respond to the offer. A lot of time and effort can thus be saved by only approaching customers with a predicted interest.

1.2 Propositional Data Mining

An important formalism in Data Mining is known as *Propositional Data Mining*. The main assumption is that each individual is represented by a fixed set of characteristics, known as *attributes*. Individuals can thus be thought of as a collection of attribute-value pairs, typically stored as a vector of values. In this representation, the central database of individuals becomes a table: rows (or *records*) correspond to individuals, and columns correspond to attributes. The algorithms in this formalism will typically employ some form of propositional logic to identify subgroups, hence the name Propositional Data Mining.

Example 1.1 Consider again the customers of a bank. Each individual can be characterised by personal information, such as *name*, *gender* and *age*, as well as by bank-related information such as *balance*, *nr. credit cards*, etc. Figure 1.1 gives an example database in tabular form. Propositional algorithms typically build models by considering subgroups identified by constraints on the propositional data, e.g. 'the group of male customers who own more than one credit card' or 'the adults'.

name	gender	age	balance	nr. credit cards
Ritchie	M	43	4,000	2
Rendell	F	12	-2,000	0
Gross	F	81	85,000	0
Ferguson	M	62	26,000	1
Smith	F	27	1,000	3

Figure 1.1 A propositional database.

Due to its straightforward structure, the Propositional Data Mining paradigm has been extremely popular, and is in fact the dominant approach to analysing a database. A wide range of techniques has been developed, many of which are available in commercial form. In terms of designing Data Mining algorithms, the propositional paradigm has a number of advantages that explain its popularity:

- Every individual has the same set of attributes. An individual may not have a value for a particular attribute (i.e. have a NULL-value), but at least it makes sense to inquire about that particular attribute. Also, each attribute only appears once, and has a single value.
- Individuals can be thought of as points in an n-dimensional space. Distance measures can be used to establish the similarity between individuals. Density estimation techniques can be used to help discover interesting regions of the space.
- Attribute values are complementary; constraints on attributes divide the individuals in complementary subgroups. This makes gathering and using statistics concerning individual attributes – an important tool in Data Mining, as we shall see later on – a straightforward operation. Also, the use of operators is simple, as complementary operators correspond to complementary subgroups (e.g. $=$ and $\neq$, or $<$ and $\geq$).

- The meta-data describing the database is simple. This meta-data is used to guide the search for interesting subgroups, which in Propositional Data Mining boils down to adding propositional expressions on the basis of available attributes.

There is a single, yet essential disadvantage to the propositional paradigm: there are fundamental limitations to the expressive power of the propositional framework. Objects in the real world often exhibit some internal structure that is hard to fit in a tabular template. Some typical situations where the representational power of Propositional Data Mining is insufficient are the following:

- Real world objects often consist of parts, differing in size and number from one object to the next. A fixed set of attributes cannot represent this variation in structure.
- Real-world objects contain parts that do not differ in size and number, but that are unordered or interchangeable. It is impossible to assign properties of parts to particular attributes of the individual without introducing some artificial and harmful ordering.
- Real-world objects can exhibit a recursive structure.

1.3 Structured Data Mining

The Propositional Data Mining paradigm has been popular because of the simple tabular structure it proposes. This property is, at the same time, its weakness. Many databases, especially of large industrial nature, are simply too complex to analyse with a propositional algorithm without ignoring important information. Rather than working with individuals that can be thought of as vectors of attribute-value data, we will have to deal with structured objects that consist of parts that may be connected in a variety of ways. Data Mining algorithms will have to consider not only attribute-value information concerning parts (which may be absent), but also important information concerning the presence of different types of parts, and how they are connected.

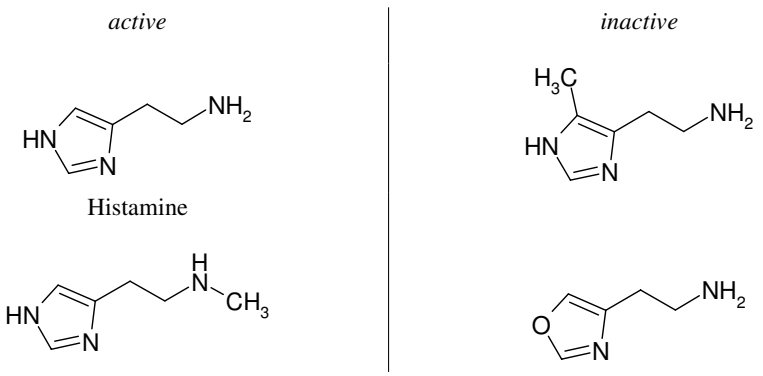


Figure 1.2 A database of Histamine-related compounds.

Example 1.2 The chemical compound Histamine is a neuro-transmitter that regulates a number of processes in the human body, such as the activity of the heart and acid secretion in the stomach [90]. Histamine and variations thereof work because they bind to specific configurations of amino acids known as Histamine H_2-receptors. For a compound to bind to the H_2-receptor (to be *active*), it will have to exhibit certain structural properties that make it fit exactly in the receptor spatially. Specifically, three nitrogen atoms are required in the same spatial arrangement as they appear in Histamine (see Figure 1.2). Furthermore, the space around the compound should not be obstructed by extra groups of atoms.

Figure 1.2 shows a database of four compounds, two of which are active, and two of which inactive. A typical Data Mining exercise would aim to explain the difference in activity based on the structural properties. Clearly, the chemicals will be poorly represented in a single table and Propositional Data Mining algorithms are difficult to apply. A more sophisticated representation and analysis scheme would consider the structure and would, for example, discover fragments of compound that can form discriminative rules. For example an HN–CH fragment occurs in all positive, and none of the negative cases.

Although a range of representations for structured data has been considered in the literature, structured individuals can conceptually be thought of as annotated graphs. Nodes in the graphs correspond to parts of the individual. A node can typically be of a class, selected from a predefined set of classes, and will have attributes associated with it. Available attributes depend on the class. For example, a molecule will consist of atoms and bonds (the classes), and each class will have a different set of attributes, such as *element* and *charge* for atoms, and *type* for bonds. The edges in the graph represent how parts are connected.
We refer to the class of techniques that support the analysis of structured objects as *Structured Data Mining*. These techniques differ from alternative techniques, notably propositional ones, in the representation of the individuals and of the discovered models. Many of the concepts of Data Mining are relatively representation-independent, and can therefore be generalised from Propositional Data Mining. For example, individuals and interesting subgroups play the same role. What is different is the definition of subgroups in terms of structural properties of the individuals. Much of the remainder of this thesis is dedicated to finding good ways of upgrading powerful concepts and techniques from Propositional Data Mining to the richer structured paradigm.
Structured Data Mining deals with a number of difficulties that translate from the advantages of Propositional Data Mining listed in the previous section:

- Individuals do not have a clear set of attributes. In fact, individuals will typically consist of parts that may be queried for certain properties, but parts may be absent, or appear several times, making it harder to specify constraints on individuals. Furthermore, it will be necessary to specify subgroups on the basis of relationships between parts, or on groups of parts.
- Individuals cannot be thought of as points in an n-dimensional space. Therefore, good distance measures cannot be defined easily.
- Complementary subgroups can *not* be obtained by simply taking complementary values for certain properties, such as attributes of parts. For example, the set of molecules that contain a C-atom and the set of molecules that contain an H-atom are not disjoint. The different values for an attribute of a part (atom) can not be used to partition the individuals (molecules).
- The meta-data describing the database is extensive. Typically, the meta-data will not only describe attributes of the different parts, but also in general terms how parts relate to each other, i.e. what type of structure can be expected. Good Structured Data Mining algorithms will use this information to effectively and efficiently traverse the search space of subgroups and models.

Over the last decade, a wide range of techniques for Structured Data Mining has been developed. These techniques fall roughly into four categories, which can be characterised by the choice of representation of the structured individuals. Although within each category the identification with the chosen representation is often very strong, it makes sense to view them in the broader perspective of Structured Data Mining. The four categories are:

- **Graph Mining** [19, 48, 79, 80] The database consists of labelled graphs, and graph matching is used to select individuals on the basis of substructures that may or may not be present.
- **Inductive Logic Programming (ILP)** [9, 11, 12, 13, 14, 22, 28, 30, 31, 70, 83, 89] The database consists of a collection of facts in first-order logic. Each fact represents a part, and individuals can be reconstructed by piecing together these facts. First-order logic (often Prolog) can be used to select subgroups.
- **Semi-Structured Data Mining** [1, 8, 16, 81, 104] The database consists of XML-documents, which describe objects in a mixture of structural and free-text information.
- **Multi-Relational Data Mining (MRDM)** [7, 31, 53, 59, 61, 62, 63, 74, 107] The database consists of a collection of tables (a relational database). Records in each table represent parts, and individuals can be reconstructed by joining over the foreign key relations between the tables. Subgroups can be defined by means of SQL or a graphical query language presented in Chapter 4.

In Chapter 2 we will examine Structured Data Mining in depth, and compare the four categories of techniques according to how they approach different aspects of structured data.

1.4 Multi-Relational Data Mining

The approach to Structured Data Mining that is the main subject of this thesis, Multi-Relational Data Mining, is inspired by the relational model [21, 100, 101]. This model presents a number of techniques to store, manipulate and retrieve complex and structured data in a database consisting of a collection of tables. It has been the dominant paradigm for industrial database applications during the last decades, and it is at the core of all major commercial database systems, commonly known as relational database management systems (RDBMS). A relational database consists of a collection of named tables, often referred to as *relations* that individually behave as the single table that is the subject of Propositional Data Mining. Data structures more complex than a single record are implemented by relating pairs of tables through so-called *foreign key relations*. Such a relation specifies how certain columns in one table can be used to look up information in corresponding columns in the other table, thus relating sets of records in the two tables.

Structured individuals (graphs) are represented in a relational database in a distributed fashion. Each part of the individual (node) appears as a single record in one of the tables. All parts of the same class for all individuals appear in the same table. By following the foreign keys (edges), different parts can be joined in order to reconstruct an individual. In our search for patterns in the relational database, we will need to query individuals for certain structural properties. Relational database theory employs two popular languages for retrieving information from a relational database: relational algebra and the Structured Query Language (SQL). The former is primarily used in the theoretical settings, whereas the latter is primarily used in practical systems. SQL is supported by all major RDBMSs. In this thesis we employ an additional (graphical) language that selects individuals on the basis of structural properties of the graphs. This language translates easily into SQL, but is preferable because manipulation of structural expressions is more intuitive.

1.5 Outline of this text

This thesis is structured as follows. The following three chapters contain an introductory text, detailing the concepts and techniques relevant for Structured Data Mining in general and Multi-Relational Data Mining in specific. Chapter 2 defines major concepts shared by all SDM paradigms, and demonstrates how each paradigm implements these concepts. Chapters 3 and 4 define in more detail how our MRDM approach works. It covers how structured individuals are represented and

queried in a relational database. Parts of these chapters were previously published in the following papers:

> Knobbe, A., Siebes, A., Blockeel, H., Van der Wallen, D. *Multi-Relational Data Mining, using UML for ILP*, In Proceedings of PKDD 2000, LNAI 1910, 2000
>
> Knobbe, A., Blockeel, H., Siebes, A., Van der Wallen, D. *Multi-Relational Data Mining,* In Proceedings of Benelearn '99, 1999

Chapter 5, Multi-Relational Rule Discovery and Chapter 6, Multi-Relational Decision Tree Induction, cover two important mining techniques in MRDM. Both techniques are based on a particular means of capturing structural features. The text in these chapters is based on the second paper mentioned above, as well as the following paper:

> Knobbe, A., Siebes, A., Van der Wallen, D. *Multi-Relational Decision Tree Induction*, In Proceedings of PKDD '99, LNAI 1704, pp. 378-383, 1999

Chapters 7 through 9 investigate an alternative way of extracting structural features, namely by means of aggregate functions. Chapter 7, Aggregate Functions, provides the necessary preliminaries for the subsequent chapters. In Chapter 8, Aggregate Functions & Propositionalisation, we present a method to flatten a multi-relational database using these aggregate functions in order to analyse the resulting table using traditional techniques. This method was previously published in the following paper:

> Knobbe, A., De Haas, M., Siebes, A. *Propositionalisation and Aggregates,* In Proceedings of PKDD 2001, LNAI 2168, pp. 277-288, 2001

Chapter 9 describes how this richer method of capturing structured features can be integrated in the rule discovery techniques introduced in Chapter 5. This method was previously published in:

> Knobbe, A., Siebes, A., Marseille, B. *Involving Aggregate Functions in Multi-Relational Search*, In Proceedings of PKDD 2002, LNAI 2431, 2002

Chapter 10, MRDM Primitives, covers how MRDM techniques can gather the necessary statistics from the database using a predefined set of queries. This subject has appeared in most of the above-mentioned papers.

Chapter 11 considers MRDM on a more methodological level. It provides a blueprint for MRDM projects, focussing on the activities that precede the actual modelling step, and presents ProSafarii, a system that supports a user in the pre-processing phase of MRDM projects.

Finally, Chapter 12 concludes with a discussion and pointers for future work.

This page intentionally left blank

2 Structured Data Mining

In this chapter we give an overview of the genus of Data Mining paradigms that deal with structured data. We start with definitions of the major concepts in Structured Data Mining having to do with how structured data is represented and how knowledge can be extracted from this data. These concepts are shared by the different SDM paradigms that are the subject of the remaining sections of this chapter. We briefly outline each approach, and describe how they implement SDM. Subsequently, we describe on a more detailed level the strengths and weaknesses of each paradigm.

2.1 Structured Data

As was outlined in Chapter 1, the central subject of analysis is the individual. We assume in Structured Data Mining that an individual consist of *parts* that are somehow connected to form a structured individual. Parts can be thought of as small portions of an individual that are atomic: they exhibit no internal structure. All structural relations are between parts, rather than within parts. Parts typically have a number of attributes associated with them. They can thus be thought of as tuples that behave similar to the flat individuals that are the subject of Propositional Data Mining. In general, structured individuals will not be arbitrary collections of parts. Parts will appear in a relatively small number of types, referred to as *classes*. All parts, over all individuals, are instances of one of the classes. A class determines which attributes are available for all instances of that class.

Definition 2.1 A *class C* is a triple (*c, S, D)* in which *c* is the class-name, *S* is the schema, a set of attributes with their domain $S = \{a_1 : D_1, \ldots, a_n : D_n\}$, and *D* is the domain of *C*: $D_C = \prod_{i=1}^{n} D_i$.

Definition 2.2 A *part* of class *C* is a tuple $(a_1, \ldots, a_n) \in D_C$.

Not just the characteristics of parts are important in Structured Data Mining, but also how they relate to form structured individuals. We will think of individuals as annotated graphs, where the nodes represent the parts. Labelled, but undirected, edges between parts represent the relationship between pairs of parts. Other than the label, the edge provides no information concerning the relationship between the parts. Typically, there will not be edges between arbitrary pairs of parts. Most data representation schemes will only allow relations between specific classes of parts to enforce certain types of structure in the individuals. Furthermore, there will often be restrictions on

the number of parts of a certain class that may be related to a part of some other class, and vice versa. A definition of the restrictions on relations between parts of two classes will be referred to as an *association* between two classes. Edges in an individual can thus be thought of as instances of an association, where the label refers to the association. It should be noted that this use of the term *association* is not related to the term *association rules*, which indicates a popular family of models of statistical dependency [33]. The present associations are hard constraints on the data.
The central input to the Structured Data Mining process is now the *database*, simply a collection of individuals. Although each individual will most probably be different in parts and structure, the set of individuals will adhere to a common set of restrictions on the appearance of individuals. This collection of restrictions is referred to as the *data model* of the database, and consists of the definitions of available classes (including attributes) of parts, plus the definitions of associations between classes.

Example 2.1 Consider a database of molecule-descriptions. Each individual describes a molecule, and consists of parts that come in three classes: a piece of information describing general properties of the molecule (**molecule**), pieces of information for each atom (**atom**), and pieces of information for each bond between two atoms (**bond**). Each of these classes has a fixed set of attributes, for example **atom** has attributes *element* and *charge*. The parts of these three classes are related to each other through four associations: one determines which atoms belong to which molecule. Similarly there is an association that determines which bond belongs to which molecule. The two remaining associations determine the two atoms involved in each bond.

Definition 2.3 A *data model* is a rooted undirected connected graph $M = (C, A)$, where:

1. C is a set of class-names
2. $A \subseteq C \times C$ such that, if $(c, d) \in A$, then $(d, c) \in A$.

The elements of A are called *associations.* The root of the graph is denoted by c_0.

Definition 2.4 An *individual* for data model $M = (C, A)$ is a rooted undirected connected graph $i = (P, E)$, where P is a set of parts, and E is a set of triples (p, q, a) such that

1. $a = (c, d) \in A$
2. p is a part of class c, and q is a part of class d
3. there is a unique part $p_0 \in P$ such that if $(p, q, (c_0, c_i)) \in E$, then $p = p_0$.

The set of all individuals for a data model M is denoted by L_M.

Definition 2.5 A *database* for data model M is a set of individuals for M.

2.2 Search

In Chapter 1 we briefly mentioned the *interesting subgroup* as an important ingredient of Data Mining algorithms. A large number of subgroups will be considered by such algorithms in order to produce a manageable set of interesting subgroups that will be the basis for higher-level models of the database. Rather than being interested in subgroups as enumerations of individuals belonging to it (extensional approach), we consider short descriptions of the properties shared by all members of the subgroup (intensional approach). We refer to such descriptions as *patterns*. In Structured Data Mining, patterns express certain required properties concerning the presence of certain parts and structural relationships between them. Patterns can thus be used to match individuals, and define subgroups.
In order to express and manipulate patterns, we require a *pattern language*. As the individuals are structured, the pattern language will need to be able to express structural properties. Because of our

definition of individuals as graphs, the language will conceptually be graphical. In fact, a number of Structured Data Mining paradigms, including the Multi-Relational Data Mining approach we present in this thesis, use variations of graphs as patterns. Other paradigms, Inductive Logic Programming and Semi-Structured Data Mining, rely on existing languages for expressing structural constraints.

Definition 2.6 A *pattern language* is a pair $L_p = (P, \textbf{covers})$ such that P is a set of expressions, and **covers** is a function $P \times L_M \rightarrow \{0, 1\}$. A pattern $p \in P$ is said to *cover* an individual $i \in L_M$ *iff* $\textbf{covers}(p, i) = 1$. If clear from the context, we will often write L_p to denote P.

Definition 2.7 A *pattern* in a pattern language $L_p = (P, C)$ is an expression $p \in P$.

Definition 2.8 A *subgroup* S_p is a set of individuals in a database D that is covered by a given pattern p: $S_p = \{i \in D \mid \textbf{covers}(p, i)\}$.

Example 2.2 Consider a database D of individuals $i = (N, E)$, and the pattern language $L_p = (P, \textbf{covers})$, where P is the set of graphs $G = (N', E')$, and $\textbf{covers}(G, i)$ *iff* there exists an injection M: $N' \rightarrow N$, such that

$$\forall e \in E' : (M(e.p), M(e.q), a) \in E$$

where a is an uninstantiated variable representing an unspecified association. This pattern language can be used to describe particular subgraphs that may or may not appear in individuals in D. The location of the root of the individuals, as well as the class and attributes of its parts are ignored. By refining P and adding more constraints on the injection M, we can define more expressive pattern languages.

Most Data Mining algorithms are characterised by an extensive search for interesting subgroups. The pattern language of choice defines a search space of patterns, which is the starting point for this search process. In order to traverse this space in a sensible, guided and efficient manner, the algorithm requires a means of judging the interestingness of a given pattern (and corresponding subgroup). In general terms we refer to such a means as a *score function*. Typically a score function considers the database and acquires statistics about the pattern at hand, which in turn produces a score. This score will help the algorithm to make informed decisions about the progress and direction of the search.
Most algorithms will not only use statistical information from the database to guide the search, but also *a priori* information about the kind of individuals that are known to exist in the database. The constraints contained in the data model concerning classes and associations tell us that we cannot expect arbitrary structures in the database. Rather, we can limit the search to patterns that are in correspondence with the data model. Restrictions on the search space based on *a priori* knowledge about the database, commonly referred to as *declarative bias*, are a very important means of keeping the search process manageable and efficient. We will consider different ways of exploiting declarative bias in Multi-Relational Data Mining throughout this thesis.

Definition 2.9 A *score function* is function f: $L_p \times 2^{L_M} \rightarrow \mathbf{R}$ such that, given a database D and two patterns p and q, $S_p = S_q \Rightarrow f(p, D) = f(q, D)$.

Definition 2.10 A *declarative bias* is a set of constraints on the patterns considered by a Data Mining algorithm.

Because we know that the search is strongly guided by the data model of the database, we can expect a certain similarity and order in the candidate patterns to be considered. Typically, a pattern will be tested on the database by means of the score function(s), and new candidate patterns will be derived from the pattern with the obtained statistics in mind. The important step of deriving new patterns by means of minimal additions to an existing pattern is known as *refinement*. We will be using a *refinement operator*, which specifies what syntactic operations are allowed on a given pattern in order to derive more complex patterns.
The predominant search mode in Data Mining is *top-down search*: start with a very general pattern, and progressively consider more specific patterns. For this, we require the refinement operator to produce subgroups that are subsets of the original subgroup. We refer to a refinement operator with this additional property as a *top-down* refinement operator.

Definition 2.11 A pattern p is *more general than* a pattern q ($p \geq q$) if all individuals covered by q are also covered by p:

$(p \geq q)$ *iff* $\forall i$: **covers**$(q, i) \rightarrow$ **covers**(p, i).

Definition 2.12 A *specialisation operator* is a function ρ: $L_p \rightarrow 2^{Lp}$ such that $\forall p \in L_p$: $\rho(p) \subseteq \{q \mid p \geq q\}$.

Definition 2.13 A *refinement operator* is a function ρ: $L_p \rightarrow 2^{Lp}$ such that, given a set of syntactic operations, $\forall p \in L_p$: $\rho(p) = \{q \mid q$ is the result of a syntactic operation on $p\}$.

Definition 2.14 A *top-down refinement operator* is a refinement operator that is also a specialisation operator.

2.3 Structured Data Mining Paradigms

The basic definitions given in the previous section outline how Structured Data Mining works in general terms. The existing Structured Data Mining paradigms as introduced in Chapter 1 all exhibit these concepts in one form or another, but we will need to consider how each paradigm implements them. This will help us translate terminology and practices from one paradigm to the other. Furthermore, by viewing paradigms as instances of a generic Data Mining approach, we will be able to distinguish the properties shared by all paradigms from those that are unique to a particular paradigm. We will start with an overview of the commonalities by considering how the different paradigms implement Structured Data Mining concepts.

Graph Mining From all four Data Mining paradigms for dealing with structured data, the Graph Mining paradigm is closest in approach to our abstract definition of structured Data Mining. Individuals are simply (labelled) graphs, and the database is hence a forest of these graphs. In most approaches a node (part) is labelled, such that each part belongs to a class. Constraints on the structure of individuals in the form of a data model are typically not present. The same graphical language used to represent individuals is used as a pattern language. Graph matching is used to determine which individuals are covered by a particular pattern.

Inductive Logic Programming (ILP) The ILP paradigm employs (small) logic programs to describe patterns. The logic programs need to be induced from a database of logical facts, hence the name Inductive Logic Programming. The facts in the database represent parts of the structured individual. The class of each part can be identified by the predicate of the fact. The notion of individual is rarely made explicit, although sometimes facts are grouped together, or keys are added

to identify the individual. A variety of declarative bias languages is used, but they mostly share the use of mode declarations [31] in order to describe how parts can be pieced together. Mode declarations restrict how certain attributes in certain predicates may be linked by shared variables in the induced logic programs. This is a slight variation on the association concept. ILP typically uses Prolog as the pattern language. ILP algorithms often allow as input not only the database of ground facts, but also intensional predicate definitions that may help the search process for interesting patterns.

Semi-Structured Data Mining (SSDM) The most recent approach to mining in structured data, Semi-Structured Data Mining deals with a database of semi-structured text, typically represented in the popular language XML [2, 17, 32]. Although semi-structured documents contain both structural information as well as fragments of free text, most approaches focus on the structural part and treat the text fragments as discrete values without internal structure. XML documents essentially represent a rooted tree, where the tree-structure is identified by markers in the text, known as *tags*, and free-text appears at the nodes (and leaves) of the tree.
The existing SSDM approaches treat documents in two technically different, but effectively similar ways. One approach is to treat a single document as the database, and regard the children of the root of the semi-structured tree as individuals. The alternative approach is to take a collection of documents as the database, and treat each document as an individual. In either way, the nodes in the tree, identified by tags, correspond to the parts, and may have several attributes.
Most XML documents come with a grammar that strongly restricts the kinds of structure that are allowed in XML documents. Typically this grammar is specified in a separate file, known as a Document Type Definition (DTD) [41], which may be shared by multiple documents. Although rarely used in SSDM, DTD is ideal as a declarative bias. In the true nature of declarative bias, a DTD not only determines when XML documents are well formed, but should be used as important information to guide and prune the search process. The existing SSDM approaches boast a variety of pattern languages in order to select individuals on the basis of features of the semi-structured tree. Most algorithms employ generic graphical or tree structures that are not specific to the technology surrounding XML. A new development is to use the XPath technology [108] that has been specifically designed for applications that need to access and query specific locations within XML documents. It can be expected that XPath, or more generally XQuery [109], will play a greater role as pattern language for SSDM in the future [16].

	MRDM	ILP	GM	SSDM
database language	relational DB	Prolog	graphs	XML
bias language	UML/ER	various	–	DTD
pattern language	SQL/SG	Prolog	graphs	graphs/XQuery

Table 2.1 Representation in Structured Data Mining approaches.

2.4 A Comparison

The general overview of approaches to mining structured data given in the previous section suggests that the paradigms mainly differ on a syntactical level. All paradigms exhibit the basic features of Structured Data Mining, albeit in a different representational framework. Table 2.1 summarises these representational differences for some important concepts. At first glance, the approaches seem interchangeable if we can simply translate between representations. However, on a more detailed level, the choice of representation does have effects on different aspects of the data and hence on the power of the mining paradigm. Below, we give a brief comparison of some of

these aspects related to the choice of representation. A summary is given in Table 2.2. It should be noted that the comparison is based on the common practices in each of the paradigms. Individual algorithms may cater to specific needs that are, however, not typical for the paradigm in question.

	MRDM	ILP	GM	SSDM
attributes	+	+	–	+
numeric values	+	+	–	–
intensional data	+	+	–	–
graph/tree	graph	graph	graph	tree
order in structural parts	–	–	–	+
structured terms	–	+	–	–

Table 2.2 Details of database representation.

Attributes ILP, SSDM and MRDM all share the notion of attributes related to parts. Attributes in all paradigms work virtually the same, although XML does not provide typing for attributes. Graph Mining is limited, as it typically does not provide for attributes at nodes in the graph. This limitation is often overcome by adding extra nodes to the graph for each attribute-value pair. This is less desirable since you loose the constraint that each attribute may have only one value.

Numeric values Attributes with numeric values typically are only treated properly in ILP and MRDM. As attributes in XML are not typed, SSDM treats numeric values as discrete values.

Intensional data Only MRDM and ILP provide for declarative definitions of (part of) the data. Intensional data is achieved in MRDM by means of view definitions: SQL queries that act as virtual tables. Intensional data is even more natural in ILP. Predicates may be defined not only by listing ground facts, but also by providing a Prolog program.

Graph/tree Only three of the SDM paradigms are able to represent individuals as full graphs. Because of its serialised nature, XML can only store tree-structured data, which restricts the power of SSDM. Admittedly, it is possible to provide links between different nodes in a tree using the ID and IDREF construct, and thus obtain more graphical structures, but this is typically not used.

Order in structural parts The textual nature of XML that causes some representational limitations also has some advantages over the other representations. Children of a parent in a tree appear in a specific order. This order may be irrelevant, which makes the parent-child relation a regular one-to-many association, but it may also be used in the analysis. Especially in sequential- or time-related domains, this may make SSDM a good choice.

Structured terms The logic programming languages used in ILP typically cater for structured terms, that is, attributes of parts may take on values that are not atomic. For example, Prolog facts may contain compound terms or lists. Although this allows for a richer representation than the alternative paradigms, structured terms are only supported by a small collection of ILP methods. Because structured terms can also be represented using more basic constructs, there is, in general, no need for this advanced functionality. In fact structured terms are sometimes used in spite of having more appropriate constructs, e.g. lists to represent sets.

	MRDM	ILP	GM	SSDM
recursion	–	+	–	–
aggregate functions	+	–	–	–
numeric data	+	–	–	–
exclusive subgraphs	–	–	+	–

Table 2.3 Expressive power of typical SDM approaches.

Aside from the representational details in which SDM paradigms may differ, the paradigms can also be characterised by the expressive power of the pattern languages and actual algorithms used. Table 2.3 gives an overview of a number of subjects in which particular paradigms excel. Again, the classification is typical, rather than absolute.

Recursion Structured individuals may contain recursive structures: several parts of the same class may be related to each other. Recursive databases are characterised by cycles in the data model. In order to capture this cyclic dependency, a pattern language needs to support recursive definitions. As Prolog includes recursion naturally, there have been a range of ILP methods that deal with recursion. As yet, ILP is unique in this respect.

Aggregate functions So-called aggregate functions are a means of characterising groups of parts in an individual by a single value. Although they are a powerful tool for building patterns, they have received virtually no attention in ILP, GM or SSDM. Aggregate functions are covered in full detail in Chapters 4, 7, 8 and 9.

Numeric data Although both MRDM and ILP deal with numeric data, its treatment in ILP is typically very static and limited, especially in comparison with analogous algorithms in Propositional Data Mining. By relying on the computational power of the RDBMS, MRDM aims to provide the same level of support for numeric data as is customary in propositional algorithms.

Exclusive subgraphs A technical, but important difference in pattern language sets GM apart from the other paradigms. Because of the nature of graph matching involved in graphical patterns, requiring two parts to be present assumes that these two parts are distinct. This is in contrast to alternative languages such as Prolog, where multiple features can map on the same part or subgraph. The exclusive subgraph matching is convenient when the cardinality of particular parts is relevant, for example when describing hands in Poker (e.g. Full House, Pairs) or functional groups in molecules (e.g. $–NH_2$).

This overview of the strengths and weaknesses of the four paradigms shows a number of things. First we can note that MRDM and ILP are relatively close in representational power. This comes as no surprise, as the analogy between relational databases and first-order logic is well-established. Still, the way both paradigms approach discovery is fairly different.
Compared to ILP and MRDM, GM seems to be rather poor. Important issues, such as numeric data and intensional definitions are mostly overlooked. Although its conceptual emphasis can be a fruitful source for new ideas in the analysis of structured data, it appears unlikely that GM can outperform the competing paradigms in industrial applications. One important exception is the natural treatment of exclusive subgraphs included in graph matching. This concept is as yet largely ignored in the other paradigms, but can be an essential tool in selected applications.
SSDM is closer to MRDM and ILP in representational power. It suffers from the limitation that individuals need to be tree-structured, which can be unrealistic. On the other hand, the strong formal basis of XML and especially the bias language DTD are an important advantage. XML is also

unique as a database language in its implicit order in parts. SSDM could be a strong candidate in domains where order is relevant, as soon as more SSDM algorithms incorporate this feature.

2.5 What's in a Name?

There has been some debate over the appropriateness of the name *Multi-Relational Data Mining*. If relational databases are concerned with data in multiple tables, why use the *multi*-prefix for mining of such databases? As is often the case, there are historical reasons for using this name. The term *Relational Data Mining* has occasionally been used in the KDD-community to describe single-table algorithms that work on data stored in relational databases. As will be clear in the remainder of this thesis Multi-Relational Data Mining is specifically concerned with the complexity stemming from having multiple tables, so the name should be clear about this.
Presently the term *Relational Data Mining* (as well as *Relational Learning*) has had some popularity among ILP researchers. This alternative name for ILP was inspired by the observation that knowledge produced by modern ILP systems can hardly be called true logic programs, and that most practical databases are of relational form rather than first-order logic. In most cases however, the *Relational Data Mining* label represents little more than an RDBMS-awareness. For example, the majority of contributions to the Relational Data Mining book [13] can be considered classical ILP in our definition. In our view, the two above-mentioned uses of *Relational Data Mining* each only satisfy one of two important hallmarks of MRDM: a focus on multiple tables, and a substantial incorporation of relational database technology.
The terms *Graph Mining* and *Semi-Structured Data Mining* describe relatively new disciplines of Data Mining. As such, they often appear more as descriptive terms, rather than well-accepted names. *Structured Data Mining* is the term we propose in this thesis for the genus of paradigms concerned with the analysis of structured data. It occasionally appears informally as *mining structured data*.

3 Multi-Relational Data

In the previous chapter, a general framework for mining structured data was outlined, and an overview was given of how the different Structured Data Mining paradigms approach such data. In this and the following chapter, we will describe in more detail how MRDM works, and introduce some basic concepts upon which the different MRDM techniques described in following chapters will build. We will explain how structured data is stored in a relational database, and how MRDM algorithms may query such data. A multi-relational pattern language will be introduced, along with a refinement operator that lays out a search space of multi-relational patterns for MRDM algorithms to explore.

3.1 Structured Data in Relational Form

Structured data will always be represented in a relational database by multiple tables. The information concerning the different parts of an individual will be distributed over these tables. There is one particular table, which we will refer to as the *target table*, that has a special role. Each record in this table corresponds to exactly one individual. The target table will be connected to other tables through foreign key relations. By following these keys the remaining data concerning individuals may be looked up. The target table will always be the starting point for searching interesting patterns, as patterns represent sets of individuals. The target table will often contain attributes that describe individuals as a whole, but may also just contain a single key-attribute that points to the structural parts in the remaining tables. Each table contains parts belonging to one particular class. All parts of this class, regardless of the individual, appear in the same table. In order to determine the individual that a part belongs to, or to collect all parts belonging to a given individual, one will have to join over the foreign key relations. Figure 3.1 and Figure 3.2 give an example of how molecules may be stored in a relational database for multi-relational analysis. Figure 3.1 demonstrates how carbon dioxide consists of six parts in three classes: one molecule,

three atoms and two bonds. In Figure 3.2 these parts are distributed over three tables that also contain data for other molecules, e.g. water.

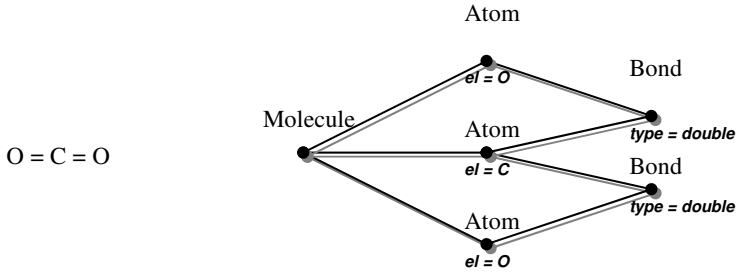


Figure 3.1 CO_2 and its graphical representation.

Molecule

id	name
1	CO_2
2	H_2O

Atom

id	mol_id	element
1	1	C
2	1	O
3	1	O
4	2	H
5	2	H
6	2	O

Bond

id	atom1	atom2	type
1	1	2	double
2	1	3	double
3	4	6	single
4	5	6	single

Figure 3.2 Relational representation of CO_2 and H_2O.

3.2 Multi-Relational Data Models

An important piece of information in Multi-Relational Data Mining is the *data model* of the database [61]. It contains a description of the structure of the database in terms of the tables and relationships between them. It provides a list of schemata of the tables: specifications of the available attributes (e.g. numeric, nominal). Furthermore, it provides information about associations between tables: specifications of how records in one table relate to records in another table. Associations can be seen as slight generalisations of the foreign key notion. They specify that two tables are related, and determine a number of details concerning the relationship, but do not fix the actual implementation of this. In practice however, an association in a relational database will always appear as a primary key-foreign key pair along with the necessary referential integrity constraint.

Associations are a bit more specific than foreign key constraints. They determine in greater detail the cardinality of the relationship between two tables, something which we will refer to as the *multiplicity* of the association. For example, we may specify that any record in one table relates to multiple, but at least one, records in another table. Such information may be very valuable to guide the Data Mining process but of less use to the integrity of the database, hence the difference between associations and foreign key constraints. Associations provide multiplicities in two directions, one for each table involved. Typically, we will be using the following four values per multiplicity:

- Zero or one (0..1)
- One (1)
- Zero or more (0..n)
- One or more (1..n)

As was proposed in [61], a visual modelling language for specifying the data model will be used, namely the Class Diagrams that are part of the Unified Modeling Language (UML) [92, 94]. This language was developed for modelling databases (as well as object oriented systems) and contains a subset of visual elements that exactly fit our needs. Figure 3.3 shows a UML Class Diagram for our molecular example.

As can be seen, there are three classes: **molecule**, **atom** and **bond**. Four associations between these classes determine how objects in each class relate to objects in another class. As a bond involves 2 atoms (in different roles) there are two associations between **atom** and **bond**. The multiplicity of an association determines how many objects in one class correspond to a single object in another class. For example, a molecule has one or more atoms, but an atom belongs to exactly one molecule.

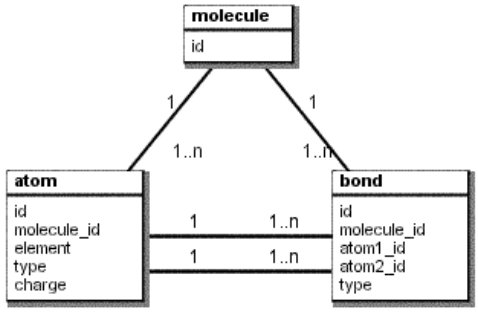


Figure 3.3 UML Class Diagram of a molecular database.

Why do we wish to use UML to express bias? First of all, as UML is an intuitive visual language, essentially consisting of annotated graphs, it is easy to write down the declarative bias for a particular domain or judge the complexity of a given data model. Another reason for using UML is its widespread use in database modelling. UML has effectively become a standard with thorough support in many commercial tools. Some tools allow the reverse engineering of a data model from a given relational database, directly using the table specifications and foreign key relations.

As attractive a visual language may be for database analysts or designers, it is less practical for machine interpretation. We have therefore defined a textual version of the necessary elements of UML Class Diagrams, in the form of an XML dialect called Multi-Relational Modelling Language (MRML) [64]. This XML format not only has applications in MRDM, but also in other database related tools such as OLAP, datawarehousing and automatic generation of (web-)applications on top of relational databases. Furthermore, due to the graphical nature of the data model, other Structured Data Mining tools can also be made to accept MRML as input. [61] demonstrates this for the ILP system Tilde. MRML contains a number of obvious elements such as tables, attributes and associations. A detailed description of the MRML format, including DTD is given in Appendix A. The following is an excerpt of the molecular database in MRML.

```
<?xml version="1.0"?>
<!DOCTYPE mrml SYSTEM "mrml.dtd">
<mrml>
     …
     <datamodel>
          <name>molecule001</name>
          <table>
               <name>atom</name>
               <attribute id="ATOMID1" type="primarykey">
                    <name>id</name>
               </attribute>
               <attribute id="ATOMID2" type="key">
                    <name>molecule_id</name>
               </attribute>
               <attribute type="nominal">
                    <name>element</name>
               </attribute>
          </table>
          <table>
               …
          </table>
          <association direction="forward">
               <name>molecule atom</name>
               <keyref id="MOLECULEID1" multiplicity="one" />
               <keyref id="ATOMID2" multiplicity="oneormore" />
          </association>
          <association direction="forward">
               …
          </association>
     </datamodel>
</mrml>
```

3.3 Structure of the Data Model

Although the information contained in the data model is generally sufficient for defining the search space of patterns considered by MRDM (or even SDM) algorithms, it often pays off to examine the structure of the data model in more detail. This examination will yield a better insight into how the search will proceed, and what results can be expected. It will also support important choices on a more procedural level, such as the definition of training and test set. Furthermore, the data model may be related to what a domain expert will know of the nature of structures present in the database. This may lead to further sources of declarative bias that are impossible to express in MRML.

3.3.1 Tables and their Roles

A good way of gaining knowledge about the structure of the data model is to consider the different roles each table can play in the representation of data. Clearly, the *target table* is an important role, played by a single table. We identify three further categories into which a table may fall. Each table belongs to one or two of the following categories:

- **Foreground-data** The foreground typically consists of tables that describe the primary domain that will be analysed. The data is gathered in-house, and is of a dynamic nature, often a snapshot of an operational system.
- **Background-data** The background, on the other hand, consists of tables that describe static knowledge about the outside world (demographics, periodic table of elements, etc.) that is not specific to the problem domain, and often of aggregated nature. It is thus the complement of the foreground.
- **Individual-data** Tables belonging to this set contain data that is individual-specific: the records in the target table define disjoint sets of records in the tables of individual-data. If a record in a table belongs to two individuals, then this table is not part of the individual-data. In order to determine whether a table belongs to the individual-data, we will have to determine the multiplicity of any of the transitive associations between this table and the target table. If this transitive association is one-to-many or many-to-many, a single record may belong to multiple individuals, and the table is not part of the individual-data.

The previously defined sets of tables influence the analysis in later stages. Let us consider the relation between foreground and individual-data. If these two sets coincide, there is a clear boundary of individuals in the data model. No dynamic data from the foreground is shared between individuals. This situation is known as the *learning from interpretations* setting in ILP [13, 31], and has a number of advantages. The most important of these is that sampling is straightforward: any division in the target table will divide the data in non-overlapping sets. This is important in later stages for internal validation, separation between training and test sets, and scalable implementations [13].

In the case where the individual-data is a proper subset of the foreground, the boundaries between individuals are unclear, and part of the data is shared. Any sampling will have to be done with greater care in order to prevent a bias on processes using samples, for example cross-validation. If the overlap has too great an influence, any tables that are part of the foreground, but not of individual-data should be left out.

Example 3.1 Figure 3.4 shows the Financial database, taken from the Discovery Challenge organised at PKDD '99 and PKDD 2000 [106]. The database is based on data from a Czech bank. It describes the operations of 5369 clients holding 4500 accounts. The data is stored in seven tables, three of which describe the activities of clients. Three further tables describe client and account information, and the remaining table contains demographic information about 77 Czech districts. We have chosen the **account** table as our target table.

The background of the Financial dataset consists of a single table: **district**. All other tables contain dynamic data that is gathered specifically for the financial domain, and thus belong to the foreground. Given the selected target table **account**, the individual-data consists of all tables except the two tables **district** and **client**. Clearly district-information is shared between many individuals (accounts). Although not directly clear from the data model, we can deduce from the existence of **disp** that there is a many-to-many relationship between accounts and clients. Therefore, clients may have multiple accounts, which excludes **client** from the individual-data.

The large proportion of the database formed by the individual-data tells us that a lot of structural data is available to distinguish among accounts. Therefore, an MRDM approach is justified. If we intend to use sampling in one form or another, we have to consider the **client** table, the only foreground table not part of the individual-data. If clients typically have multiple accounts, we may have to sample on the clients and project this sampling onto the accounts, in order to arrive at a true partitioning. Alternatively, **client** may be excluded from the analysis.

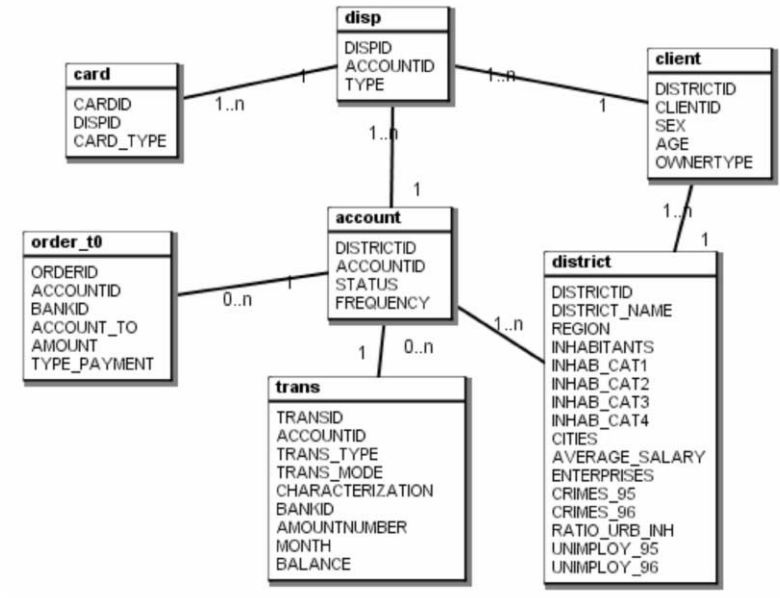


Figure 3.4 The Financial database.

It should be noted that intensional tables (views) do not necessarily belong to the background. Although the *definition* of these tables is often based on common, static knowledge about the world, the actual data represented by the intensional table can just as well be foreground. This is because it can be derived from the dynamic data in foreground tables, and can be considered a reformulation of this data.

Note also that alternative definitions of the background (and hence the foreground) have been used, particularly in the ILP field. The following uses have appeared in the literature:

- The background consists of every table (extensional or intensional) other than the target table.
- The background consists of every table that is not individual-specific (foreground equals individual-data). In the *learning from interpretations* setting this coincides with our own definition.
- The background consists of all intensionally defined tables (i.e. views or predicate definitions).

3.3.2 Directions

In general Multi-Relational Data Mining algorithms will traverse the graph of tables and associations, which makes the data model a strong way of guiding the search process. In many cases however, the data model is too general and not all paths are desirable. Data models may contain redundant associations that are useful for describing structural information in alternative ways, but harm the search process if the redundancy is not made explicit.

Example 3.2 Consider the molecular database in Figure 3.3. The data model provides two ways of expressing which molecule a bond belongs to: directly through the association between **molecule** and **bond**, and via the two atoms it binds. Although none of the bonds in the database will be

associated with another molecule than the molecule its two atoms belong to, the data model does not exclude this. Therefore, the data model suggests patterns that are known to be impossible, and MDRM algorithms will waste time considering these patterns. The alternative paths in a data model should be explored during the search, but never together in the same pattern.

Directed associations are a good way to specify this extra declarative bias, as well as to further control the produced results. If an association between two tables is directed, the search process can only introduce information in one table after the other table is introduced, in the direction of the association and not vice-versa. Directions can best be added in the following cases:

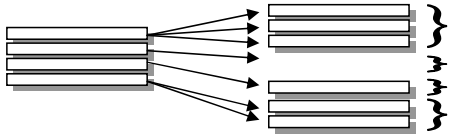
- redundant associations. An association is redundant if it represents a relation between two tables that is also represented by several other associations combined. The redundancy in Example 3.2 is solved by adding directions away from **molecule**.
- from foreground to background. In most cases it does not make sense to add structural information to the static information in the background, hence the directional bias. In the Financial dataset, we add directions to the associations connecting the demographic information in **district**. As a further advantage, this will remove the only cycle in the data model.

This page intentionally left blank

4 Multi-Relational Patterns

4.1 Local Structure

It has become clear that individuals, and even more importantly *groups* of individuals, are the main subject of analysis in MRDM. Now that we know how individuals are represented in relational databases, we should consider how to query databases of individuals, and select subgroups that share certain structural characteristics. In our MRDM framework we will mainly express such structural characteristics by building on conditions at the low level of relationships between pairs of tables. For example we will select candidate subgroups of molecules by extracting structural features from the **molecule** and the **atom** table and the association between them. We will refer to such incomplete and localised information as *local structure*. By combining conditions on the local structure (not necessarily involving the target table) we end up with expressions about individuals.
Local structure involves data in two tables, but more importantly, the association between them. The association will typically be one-to-many (or many-to-one depending on the order of the two tables). Such a one-to-many relationship basically defines a grouping on the records in the table on the 'many' side, as demonstrated below:



We can select records in one table on the basis of features of the associated group of records in the other table. This way of extracting information about local structure by means of describing groups of records will be the primary approach to building descriptions of individuals.
There are two important ways of expressing conditions on groups in local structure. Choosing one of these ways has a major impact on the pattern language to be used, and thus on the type of knowledge that can be extracted from the database. Furthermore, this choice determines the

applicable algorithms. In this thesis, we dedicate a number of chapters to each approach. These approaches are:

- Conditions on the occurrence within the group of records with a particular characteristic. These so-called *existential features* typically involve one or more attributes of the records, but can also just refer to the existence of a record, or of records that are again related to records in the other tables through other associations. This approach has been the predominant mode of work in other Structured Data Mining frameworks, such as Inductive Logic Programming and Graph Mining. In this introductory chapter and the following two chapters, we will limit the discussion to this approach.
- Conditions on the group of records as a whole. By means of so-called *aggregate functions* we can describe a group in a variety of ways. For example, groups can be selected on the basis of their size or on the average value within the group of a particular numeric attribute. Such conditions are a superset of the existential conditions, as we can include the *exists* aggregate function. The approach based on aggregate functions has only recently been receiving attention in the field of Structured Data Mining [38, 59, 63, 68, 69, 84, 85]. Chapters 7 through 9 cover the possibilities and limitations of aggregate functions.

4.2 Pattern language

If we want to select subgroups of individuals on the basis of structural properties they possess, we need to define a language that is capable of expressing existential conditions. We will be using the *selection graph* language to express multi-relational patterns:

Definition 4.1: A *selection graph* G is a pair (N, E), where N is a set of pairs (t, C) called *selection nodes*, t is a table in the data model and C is a, possibly empty, set of conditions on attributes in t of type $(t.a\ \theta\ c)$. θ is one of the following operators, $=, \leq, \geq$.
E is a set of triples (p, q, a) called *selection edges*, where p and q are selection nodes and a is an association between $p.t$ and $q.t$ in the data model. The selection graph contains at least one node n_0 that corresponds to the target table t_0.

Definition 4.2 An *empty selection graph* is a selection graph that contains no edges or conditions: $(\{(t_0, \varnothing)\}, \varnothing)$.

Definition 4.3 Let $i = (P, E)$ be an individual and $G = (N, E')$ be a selection graph. G *covers* i (**covers**(G, i)) *iff* there exists a function $M: N \rightarrow P$ such that

1. $M(n_0) = p_0$
2. $\forall n \in N$: $M(n)$ satisfies the set of conditions $n.C$
3. $\forall e \in E'$: $(M(e.p), M(e.q), e.a) \in E$

Note that although the **covers** relation is concerned with the occurrence of particular graphical structures in individuals, it is not the same as the graph matching employed in Graph Mining. For selection graphs to act as subgraphs of individuals, we would have to add the extra constraint that the function M is an injection: $n \neq n'$ implies $M(n) \neq M(n')$.

Definition 4.4 The *selection graph language SG* is a pattern language $SG = (P, \textbf{covers})$, where P is the set of possible selection graphs.

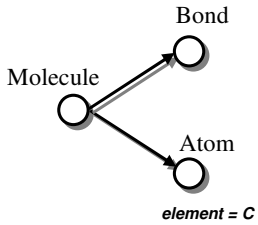
Selection graphs represent selections of individuals. The selection node n represents a selection of records in the corresponding table $n.t$ that is determined by the set of conditions $n.C$ and the

relationship with records in other tables characterised by selection edges connected to *n*. Selection graphs are more intuitive than expressions in SQL or Prolog, because they reflect the structure of the data model, and refinements to existing graphs may be defined in terms of additions of edges and nodes.
A selection graph can be translated into SQL or Prolog in a straightforward manner. The following algorithm shows the translation into SQL. It will produce a list of tables *table_list*, a list of join conditions *join_list*, and a list of conditions *condition_list*, and combine these to produce a SQL-statement. A similar translation to Prolog can be made.

```
TranslateSelectionGraph(selection graph G)

    table_list = ''
    condition_list = ''
    join_list = ''
    for each node i in G.N do
        table_list.add(i.table_name + ' T' + i)
        for each condition c in i do
            condition_list.add('T' + i + '.' + c)
    for each edge e in G.E do
        join_list.add(e.left_node + '.' + e.left_attribute + ' = ' +
            e.right_node + '.' + e.right_attribute)
    return 'SELECT DISTINCT T0. ' + t0.primary_key +
        ' FROM ' + table_list + ' WHERE ' + join_list + ' AND ' + condition_list
```

Example 4.1 The following selection graph, and its corresponding SQL statement, represents the set of molecules that have a bond, and a C-atom.



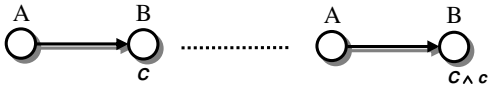

```
SELECT DISTINCT T0.id
FROM molecule T0, bond T1, atom T2
WHERE T0.id = T1.molecule_id and T0.id = T2.molecule_id
AND T2.element = 'C'
```

4.2.1 Refinements

The selection graph language provides a search space of multi-relational patterns for MRDM algorithms to explore. The MRDM algorithms in this thesis all traverse this search space in a top-down fashion: starting with a simple, very general pattern, progressively consider more complex and specific selection graphs. A refinement operator defines in more detail how this top-down traversal works.

We introduce the refinement operator ρ_{SG} for selection graphs. This refinement operator consists of two possible operations on the selection graphs, as follows:

- **Condition refinement.** This operation adds a simple condition on an attribute to a particular node in the graph. Depending on the type of the attribute, the operators ≤ and ≥ (numeric), or = (nominal) are used.



- **Association refinement.** This operation takes an association in the data model, and adds a corresponding selection edge and selection node to the graph. The new node contains no conditions.

Formally we define ρ_{SG} as:

Definition 4.5 ρ_{SG} is a refinement operator SG $\rightarrow 2^{SG}$ as follows: Let $G = (N, E) \in$ SG, $n = (t, C) \in N$, then

1. if a is a numeric attribute in t and $v \in D_a$ a value in its domain, let $c_1 = (a \leq v)$, $c_2 = (a \geq v)$, $C_1 = C \cup \{c_1\}$, $C_2 = C \cup \{c_2\}$, $N_1 = (t, C_1)$, $N_2 = (t, C_2)$, then (N_1, E) and (N_2, E) are in $\rho_{SG}(G)$.
2. if a is a nominal attribute in t and $v \in D_a$ a value in its domain, let $c = (a = v)$, $C_1 = C \cup \{c\}$, $N_1 = (t, C_1)$, then (N_1, E) is in $\rho_{SG}(G)$.
3. if $l = (t, s)$ is an association in the data model, let $n_1 = (s, \varnothing)$, $N_1 = N \cup \{n_1\}$, $e = (n, n_1, l)$, $E_1 = E \cup \{e\}$, then (N_1, E_1) is in $\rho_{SG}(G)$.

Proposition 4.1 ρ_{SG} is a top-down refinement operator.

Proof Let $p, q \in$ SG such that $q \in \rho_{SG}(p)$. Let $i \in D$ such that **covers**(q, i).
We first prove that **covers**(p, i) holds for condition refinements. Let B: $q.N \rightarrow p.N$ a bijection between selection nodes in q and p as implied by ρ_{SG} (Definition 4.5 1. and 2.). There is a single $n_q \in q.N$ such that

$$B(n_q).C \subset n_q.C.$$

Consider a function M: $q.N \rightarrow i$ as specified by definition 4.3. $n_q.C$ holds for $M(n_q)$, and hence $B(n_q).C$ holds for $M(n_q)$. We can now define a function M': $p.N \rightarrow i$ as follows:

$$M'(n) = M(B^{-1}(n)) \text{ for } n \in p.N.$$

M' satisfies definition 4.3 and hence **covers**(p, i).
We next prove that **covers**(p, i) holds for association refinements. According to Definition 4.5 3., $p.N \subset q.N$ and $p.E \subset q.E$. This means that none of the conditions for the function M as specified by Definition 4.3 is violated, and hence **covers**(p, i).
Hence ρ_{SG} is a specialisation operator, and consequently a top-down refinement operator.

□

It should be noted that only tree-structured selection graphs can be derived from the empty selection graph. The refinement operator will never produce a cycle in the graph. As such, ρ_{SG} is not *complete*: some patterns in SG cannot be derived. Although a single refinement can be added to make ρ_{SG} complete, we have chosen to restrict ourselves to tree-structured graphs. Later extensions of SG will make cycles troublesome (see Chapters 6, 9 and 10).

4.2.2 Pattern Complexity

In the remainder of this thesis, we will be presenting a number of MRDM techniques that manipulate selection graphs. In order to judge the power and limitations of these techniques, we will consider the complexity of the multi-relational patterns they are able to discover. The following measures of complexity for selection graphs will be helpful:

Definition 4.6

1. association-depth: $d_a(G)$ equals the length of the longest path starting at the root-node in G.
2. refinement-depth: $d_r(G)$ equals the sum of the number of edges and conditions in G.
3. association-width: $w_a(G)$ equals the largest sum of the number of conditions and children per node, not including the root-node.

The intuition of these definitions is as follows. An algorithm searches *association-deep* if pieces of discovered substructure are refined by adding more substructure, which results in chains of edges in selection graphs. With each new association, information from a new table is involved. An algorithm searches *refinement-deep* if it considers very specific patterns, regardless of the number of tables involved. The refinement-depth of a selection graph corresponds to the number of consecutive refinements necessary to derive the graph from the empty graph. Even propositional algorithms may produce refinement-deep patterns that contain many conditions at the root-node and no other nodes. Rather than long chains of associations, *association-wide* algorithms are concerned with the frequent reuse of a single association. If information from a new table is included, it will be further refined by extra restrictions, either through conditions on this information or through further local structure.

Example 4.2 The measures produce the following complexity characteristics for the selection graph presented in Example 4.1:

$$d_a(G) = 1,\ d_r(G) = 3,\ w_a(G) = 1$$

4.3 Characteristics of Multi-Relational Patterns

Analysing local structure by means of existential features over groups has a number of characteristics that set it apart from what one might be used to in Propositional Data Mining. These peculiarities of MRDM are important because of their impact on the design of MRDM algorithms. They are also important when interpreting correctly the output of algorithms. We summarise the most crucial ones. Note that these characteristics derive directly from predicate logic:

- Refinements may not always produce patterns with a lower coverage. This is especially true for association refinements, where nodes that are totally redundant may be added to a selection graph. This redundancy is the result of declarative bias that indicates that there will always be a part corresponding to the new node. In our molecular example, simply adding a new atom-node through the association between **atom** and **molecule** does not logically change the pattern at hand, as all molecules must contain atoms. Although such irrelevant

nodes produce no immediate benefit, they are essential because they may be the starting point for further refinements.

- Associations will often appear multiple times as edges from a single node in a selection graph. This indicates that parts with different characteristics, but of the same class, are required to appear. These different sets of characteristics need not be mutually exclusive. They may appear in a single part. Multiple edges of the same association only make sense (and should be allowed as refinements) if the data model indicates a one-to-many relationship.
- In Data Mining, it is often desirable to split a particular subgroup into mutually exclusive subgroups, for example to build hierarchical models of the data, such as decision trees. In Propositional Data Mining, this can be done for example on the basis of the values of a nominal attribute. Each value represents a subgroup, and the union of subgroups thus formed represents the original subgroup. There is no overlap between subgroups, so the size of the original subgroup equals the sum of the sizes of the splits. This simple and attractive feature does not hold for splits on nominal attributes in MRDM. Due to the one-to-many associations and the existential nature of conditions, individuals will often exhibit multiple values for the nominal attribute at the same time. This will place individuals in multiple subgroups represented by the nominal values, causing overlap, and thus problems with adding sizes. In general, nominal attributes can only be used to form true splits if they occur in the target table. As a resulting effect, it is often not possible to use the distribution of values in the nominal attribute to predict the size of subgroups. For example, if we list all atoms in a database of organic compounds, we notice that hydrogen atoms occur more often than carbon atoms. However, if we use these values to produce subgroups, we notice that both subgroups are equal. In fact, all organic molecules (in this case) contain both carbon and hydrogen.
- A similar effect occurs with numeric splits. The intuition from PDM is that as one changes the numeric threshold to increasing values as they occur in the numeric attributes, the size of the resulting subgroup will either monotonically increase or decrease. In MRDM however, often only a few of the encountered values will actually be proper thresholds. Of all the numeric values occurring within a single individual, only the minimum and maximum are valid as thresholds to in- or exclude the individual. Handling numeric data is covered in Section 4.4.
- As a corollary of the previous two effects, it should be noted that complementary operators, such as = and $\neq$, or < and $\geq$, do not generally produce complementary subgroups. This is clear if we think of the two operators as representing the two values of a binary attribute, which is a special case of a nominal attribute. Each operator should be considered independently. For numeric attributes, each inequality operator should be treated separately, together with separate sets of valid thresholds. See again Section 4.4.
- The most obscure peculiarity of MRDM has to do with interactions between multiple conditions. An interesting effect of this is that two conditions may be irrelevant in isolation, whereas combined they do produce a proper refinement. This is demonstrated by the database in Figure 4.1. The database describes two hands of playing cards. If we query this database for *suit* = ♠, we obtain two hands. Two hands are also obtained if queried for *rank* = A. We would expect the conjunction of these conditions to produce the same result, as both conditions appear not to be selective. However, there is only a single hand with *suit* = ♠ $\wedge$ *rank* = A. This example shows how a refinement may change (beneficially or not) the usefulness of a prior refinement, thus producing interactions between conditions. One way to understand this is to look at conditions not just as selections on the level of individuals,

but also on the level of parts within the individual. This change in local structure produces the described effect. Especially with numeric attributes this may cause surprising effects. As was noted before, there is a limited set of proper numeric thresholds, which is a subset of the available values. Adding new refinements may have an influence on this set of thresholds.

Hand

id
1
2

Card

hand_id	suit	rank
1	♠	A
2	♥	A
2	♠	7

Figure 4.1 Two hands of playing cards.

4.4 Numeric Data

Proper handling of numeric data in MRDM requires more attention than in Propositional Data Mining. Although the selection graph language naturally incorporates numeric conditions, extra care is required in order to obtain optimal, or even correct, results. The complications are the result of the potential non-determinate relation between individuals and numeric values appearing in tables other than the target table. Because of this relation, the distribution of values in the numeric attribute can only be used indirectly to produce thresholds for defining subgroups. Individuals typically contain multiple numeric values, only a few of which are discriminative for this individual. Of the many values appearing in the numeric attribute, only few can therefore act as candidate thresholds. Hence the distribution of all numeric values is irrelevant, and any procedure that relies on this should be considered with care. Rather than working with this distribution, we will be working with the minimum and maximum value per individual in order to find good thresholds on parts of individuals, as suggested by the following lemma:

Lemma 4.1 Let B be a bag of real numbers, and t some real, then

$$\max(B) \geq t \textit{ iff } \exists v \in B: v \geq t,$$
$$\min(B) \leq t \textit{ iff } \exists v \in B: v \leq t.$$

As Lemma 4.1 shows, only the largest and smallest values within each individual are relevant to include or exclude an individual on the basis of a numeric test. Only these values will therefore be candidates for numeric thresholds.

Example 4.3 Consider the **account** and **transaction** table in the Financial dataset (Example 3.1). Every individual (account) consists of a large number of transactions. Although the distribution of values for the numeric attributes *amount* and *balance* in the **transaction** table is interesting, not every occurring value can act as a threshold to include or exclude individuals. Figure 4.2 presents the transactions of two individuals (black and grey dots respectively). To include an individual on the basis of an existential test involving one of these attributes, only the minimum and maximum value per individual is relevant. These thresholds are represented by the dashed lines in the graph.

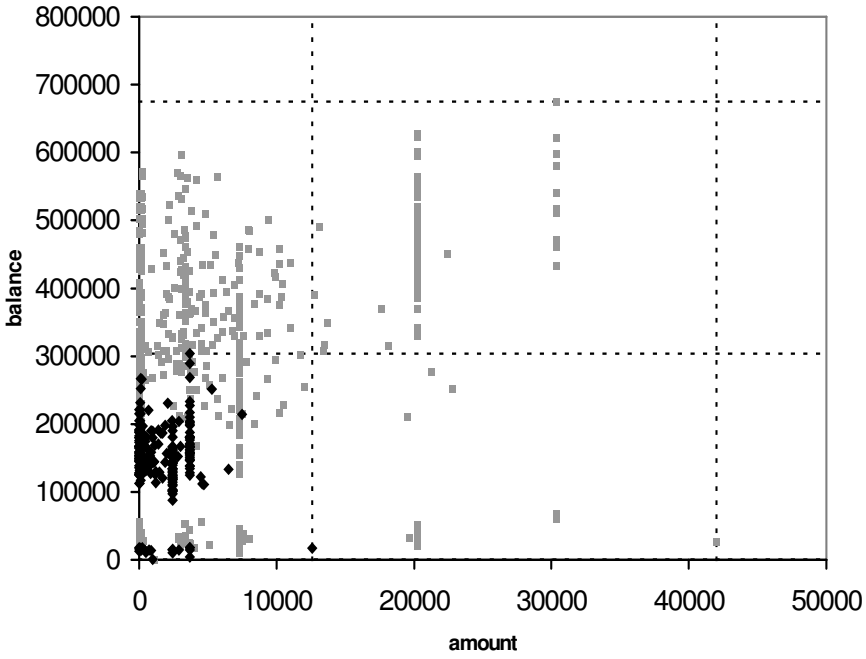


Figure 4.2 The amount and balance of transactions related to two individuals in the Financial database. Dashed lines represent the minimum and maximum values per individual.

Lemma 4.1 furthermore demonstrates that there is a difference between the set of thresholds appropriate for the ≤ and the ≥ operator. This means that any procedure that selects thresholds will have to be performed separately for each operator. Figure 4.3 shows the thresholds for ≤ (minimum) and ≥ (maximum), extracted from the **account** and **transaction** table (Example 4.3). Dots represent the bottom left, and top right corners of the two-dimensional bounding box of transactions per individual. Clearly, there is a great difference between applicable thresholds for ≤ and ≥.

As a further corollary of Lemma 4.1, note that we cannot assume that two subgroups identified by complementary conditions are complementary.

4.4.1 Discretisation

Proper treatment of numeric data has traditionally received little attention in Structured Data Mining. Only a small number of ILP and MRDM tools support full treatment of numeric data in the style of the majority of propositional tools. If one intends to deploy an algorithm that does not handle numeric data, global discretisation is the obvious solution. This entails pre-processing all numeric attributes into nominal equivalents based on fixed numeric thresholds, as opposed to allowing the algorithm to generate optimal thresholds dynamically and in a context-sensitive manner (local discretisation). As fixing thresholds prior to analysing has its limitations and can

introduce serious loss of information, global discretisation should only be considered as a work-around.

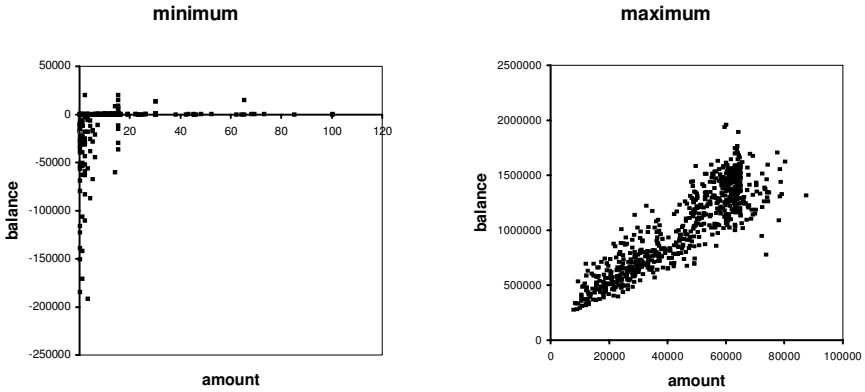


Figure 4.3 Difference in distribution for the minimum and maximum values of amount and balance per individual.

There are two major reasons for applying (global) discretisation:

- Efficiency. Especially real-valued attributes tend to have a very high cardinality, which may result in unacceptable numbers of hypotheses to be tested. Fine discretisation, typically on the order of 100 to 1000 thresholds, may remove this problem while only slightly affecting the accuracy. Note that this only works well if the subsequent analysis is performed by an algorithm that does support full treatment of numeric data.
- Limitations of the algorithm. If an algorithm can only deal with nominal data, coarse discretisation is necessary. Such a pre-processing step typically involves a very small number of thresholds, and can be seen as a form of clustering.

The above observations hold for both propositional and multi-relational data. With discretisation in MRDM however, we need to take into account the complications mentioned in the previous section. To overcome the distribution-related problem, Van Laer [102] suggests weighting each value with the inverse of the number of values associated with the related individual. We propose an alternative method that selects thresholds by taking the minimum and maximum per individual, and then applying regular discretisation procedures.

The predominant approach in discretisation has been to take the generated set of n thresholds to translate the numeric attribute into a nominal attribute of cardinality $n + 1$ (e.g. low, medium, high). Such a representation is very limiting, as it leaves no room for Data Mining algorithms to exploit the inherent order in values or consider selections of varying coverage by repeated refinements on the numeric domain. Instead, we propose a representation of the thresholds by n binary attributes. By adding such attributes as conjuncts to the hypothesis through repeated refinements, a range of intervals can be considered. A further advantage of this representation is that the accuracy is less sensitive to the number of thresholds as the size of the intervals does not decrease with the number of thresholds.

As noted before, the inequalities $<$ and $\geq$ are not complementary in MRDM. The process of selecting thresholds and introducing new binary attributes should therefore be performed twice, once for each inequality test.

Example 4.4 Consider the **order** table in the Financial dataset. The *amount* attribute can be discretised with the respect to the target table **account** as follows. For the ≤ operator we can introduce three thresholds (1000, 2800 and 4720) which divide the accounts up into four subgroups of roughly equal coverage. These thresholds produce three new attributes in **order** (*amountlow1* = (*amount* ≤ 1000) etc.). This process is repeated with thresholds 7820, 5850 and 4000 for the ≥ operator, to form attributes *amounthigh1* etc. These thresholds produce a similar equal-coverage division, although generally not with the same subgroups. Note how MRDM algorithms can consider subgroups of 75%, 50% and 25% of the accounts in two ways, or go through these subgroups in progressive refinements by adding conditions on the binary attributes.

5 Multi-Relational Rule Discovery

In this chapter, we take a common Data Mining task, known as *Rule Discovery*, and show how it can be approached in a multi-relational setting. Rule discovery is concerned with producing a collection of regularities from a given database. As the name suggests, the regularities take the form of rules. In this case, we present an algorithm that finds rules with a previously specified conclusion, as follows:

$$S \rightarrow T$$

where S is an interesting subgroup identified by a selection graph, and T is a subgroup identified by a simple propositional condition on the target table that is fixed (the *target*). Rules are local models: they specify some knowledge about the individuals covered by S, but not about the remainder of the database. The rules that are reported are independent. They simply express a regularity of sufficient interest, regardless of other rules that may express (partially) similar results. As such, the collection of rules should not be treated as a classifier: not all conceivable individuals may be covered, and some individuals may be covered by multiple (competing) rules.

5.1 Rule Discovery

Our rule discovery algorithm follows the generic Data Mining algorithm outline, as proposed by Hand, Mannila and Smyth [43]. This generic algorithm is centred around a top-down search algorithm with five variables:

- The data mining *task* that is addressed.
- The *structure* (functional form) of the model considered.
- The *score function* used to judge candidate models.
- The *search strategy* employed to traverse the space of models.
- The *data management technique* used for storing, indexing and retrieving data.

In our algorithm there are only three variables as the data mining task is Rule Discovery, and the functional form is fixed to the above-mentioned rules. The remaining three variables are specified by the user, along with other details such as the data model and the target concept.
Along the *score function* dimension, we are using the collection of rule evaluation measures proposed by Lavrač, Flach and Zupan [73]. Their publication considers a number of measures, along with the intuition behind their definition. They show that a number of these are equivalent. The condensed list of rule evaluation measures supported by our algorithm is thus as follows:

$$\text{support}(S \rightarrow T) = P(ST)$$
$$\text{coverage}(S \rightarrow T) = P(S)$$
$$\text{accuracy}(S \rightarrow T) = P(T \mid S)$$
$$\text{specificity}(S \rightarrow T) = P(\neg S \mid \neg T)$$
$$\text{sensitivity}(S \rightarrow T) = P(S \mid T)$$
$$\text{novelty}(S \rightarrow T) = P(ST) - P(S) \cdot P(T)$$

In practice we will often choose accuracy or novelty, as they both express what is close to an expert's judgment of "interesting" [73]. Accuracy, known as *confidence* in association rule mining, specifies what fraction of the covered individuals actually belongs to the target. Novelty measures to what extent co-occurrence of S and T differs from what could be expected had S and T been independent. Another interpretation of novelty is *weighted relative accuracy*: rules are novel if they have a relatively high accuracy, combined with a relatively high coverage. Novelty thus overcomes an important drawback of accuracy, which is to accept overly specific rules.
Along the *search strategy* dimension, the algorithm provides four options:

- Exhaustive search using breadth-first search (BFS).
- Exhaustive search using depth-first search (DFS).
- Best-first search.
- Beam search.

Notice that of these four, the last two are more practical for larger and realistic databases, due to their heuristic nature. All strategies are implemented by keeping a priority queue of candidate patterns to consider. The way the priority is computed for a new candidate simply determines the resulting strategy. BFS and DFS simply place candidates at the end and at the beginning of the queue, respectively. Best-first search uses the computed value of the evaluation measure at hand as the priority, whereas beam search combines the search-depth and the computed value.
The *data management technique* dimension allows different means of actually obtaining the necessary counts for computing the score function from the database. By defining a set of standard queries, independence between the mining algorithm and the data access is achieved. Section 5.2 demonstrates this in more detail.
The basic algorithm for rule discovery is shown in Figure 5.1. The algorithm basically continues considering candidate rules in the order specified by the search strategy, until all available candidates have been exhausted. Consideration of a candidate may lead to new refinements that are subsequently added to the priority queue. Actual access to the database is restricted to the procedure that computes the evaluation measure for the refinement at hand (*computeMeasure*). With databases of industrial size, this is where the bulk of the computational effort lies. It is good to note that all of the multi-relational issues are hidden in the different procedures, such as enumerating refinements and evaluating rules. In fact, the algorithm is representation-independent. In the next section, it will become apparent that, in practical implementations, it is necessary to deal with issues related to the multi-relational representation of data, for example, to achieve efficiency.

SGRules(Data model M, Database D, integer d_{max}, float v_{min})

```
Q = {M.empty()}
while Q ≠ Ø
        c = Q.extractMax()
        for each refinement r ∈ ρ_SG(c)
                v = D.computeMeasure(r)
                if r.depth() < d_max
                        Q.insert(r, v)
                if v ≥ v_min
                        R.insert(r)
Return R
```

Figure 5.1 The SGRules algorithm.

5.2 Implementation

The basic algorithm for multi-relational rule discovery, as outlined in the previous section, demonstrates the primary concepts involved in an MRDM algorithm. For a truly practical implementation however, a number of details will need to be dealt with, especially where efficiency and scalability are concerned. This section presents such an implementation.

The commercial Data Mining package Safarii [93] provides an implementation of multi-relational rule discovery. The package boasts a Client/Server architecture with a clear separation between the relatively lightweight search process and the computation-intensive evaluation of candidate patterns in the database. This separation is attractive from an architectural point of view because it allows a variety of RDBMSs to be used, ranging from well-known commercial database systems to Data Mining-optimised query engines such as Monet [15], without having to alter software on the client side. Furthermore, the client and server can be run on separate workstations, each optimised for the specific workload, thus making a first, but essential, step towards scalability.

The separation of search process and evaluation of candidate patterns is achieved through the definition of so-called multi-relational data mining primitives: small data structures of statistical information concerning some aspects of the contents of the database. Data mining primitives typically contain sufficient information to determine the evaluation measure value for a range of similar candidate patterns. The mining algorithm never accesses data directly, only through the use of a small set of predefined primitives. The statistical summaries can generally be produced by a single query to the database, for example a single SQL-statement. As a result, the RDBMS can optimise the access to the data that is required for evaluating multiple patterns, in one single process. Furthermore, it leaves some questions concerning details about exactly which patterns to evaluate to the RDBMS.

Multi-relational data mining primitives are covered in detail in Chapter 10, but we briefly demonstrate their benefits here. Consider a good example of a molecular database: the Mutagenesis database [96], which describes a set of 188 molecules falling in two classes, *mutagenic* and *non-mutagenic*. Assume that we are considering the multi-relational pattern ‘molecules that contain an atom’. We have already obtained counts for the positive (i.e. mutagenic = T) and the negative examples, 125 and 63 respectively. We can now run a primitive known as NominalCrossTable, in order to obtain information concerning the presence of atoms of a specific element, and the effect this presence may have on the mutagenicity. The primitive in question can be expressed in SQL as follows. It basically sums up the available elements and counts the positive and negative examples in which each element occurs.

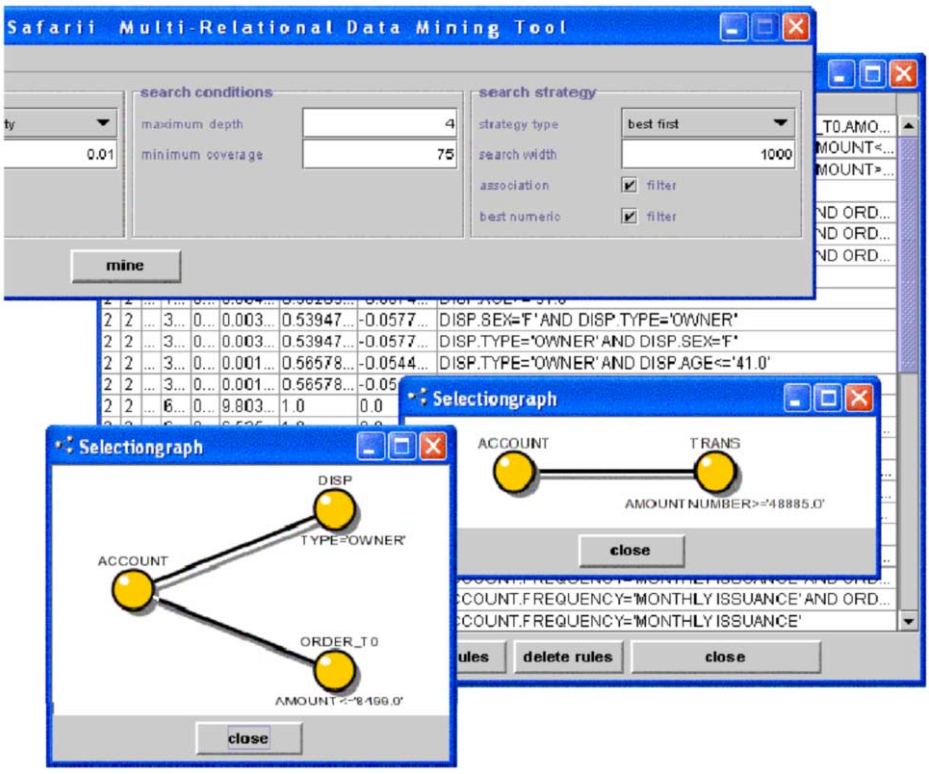


Figure 5.2 The Safarii MRDM system.

```
SELECT molecule.mutagenic, atom.element,
     COUNT(distinct molecule.id)
FROM molecule, atom
WHERE molecule.id = atom.molecule_id
GROUP BY molecule.mutagenic, atom.element
```

Figure 5.3 shows the result of this query. The statistical information obtained tells us that a condition refinement involving `atom.element` is of limited use in the present context. The elements carbon, hydrogen, nitrogen and oxygen (C, H, N, O) appear in all individuals, and the remaining elements are infrequent, producing rules of low support. Chlorine may be of some interest, as it appears in 12.7% of the negative examples, compared to 2.4% for the positive examples. A rule involving Cl has a slight (negative) novelty for a rule predicting mutagenic = T:

$$P(ST) - P(S)\cdot P(T) = \frac{3}{188} - \frac{11}{188}\cdot\frac{125}{188} \approx -0.023$$

Rules with a negative novelty indicate that a rule with the inverse conclusion is novel. Note that it does not make sense to add up counts over the elements, as molecules can contain different elements. It does, however, make sense to add counts over the target attribute, as this appears in the

target table. Furthermore, note how the primitive only reports combinations that actually occur in the database for the pattern at hand. For example, it does not consider the 100 odd remaining elements of the periodic table.

	Br	C	Cl	F	H	I	N	O
T	1	125	3	4	125	1	125	125
F	1	63	8	5	63		63	63

Figure 5.3 A NominalCrossTable primitive.

Next to deciding on a proper architecture, a practical implementation also needs to deal with a variety of details that may influence the efficiency or quality of results obtained. The following choices were made in the design of Safarii:

- Condition refinements involving numeric attributes will often give rise to a large number of very similar patterns, each of which may be the starting point for a very lengthy search process. This means that an unacceptable amount of computation may be spent on considering lots of slight variations. More importantly, the set of resulting rules will contain many copies of rules that only differ in a single numeric threshold. As a solution, only the condition refinement with the optimal numeric threshold will be reported and added to the queue of candidates.
- Association refinements do not necessarily lead to a reduced coverage. This is true in particular for refinements over associations that have a one or one-or-more multiplicity (e.g. a molecule has at least one atom). Therefore, interesting rules can give rise to rules of equal interest, containing irrelevant nodes. Although such rules are essential for further refinement of the irrelevant nodes, they should not be reported.
- Irrelevant nodes can give rise to overly complex queries to the database. When queries are produced from refinements, they should be filtered for irrelevant joins.

5.3 Experiments

As a demonstration of how Safarii works, we perform a number of experiments on the previously mentioned Financial database. Although this is a well known database, few results under comparable conditions have been published, making quantitative comparisons with competing tools difficult. Rather, we show Safarii's performance under varying circumstances to demonstrate the kind of results that can be obtained and the computational effort involved.

The experiments were performed on a standard desktop PC (CPU 1 GHz, 256 Mb main memory) using a MySQL database (64 Mb key-buffer, 4 Mb sort-buffer) running on the same machine. The algorithm was run with a refinement-depth of 5, a candidate queue of 30, and best-first search. These moderate settings produce reasonable results. A more elaborate search may produce slightly better results, but at the price of substantially increased running time.

As individuals, we consider the accounts that have a loan, and define a target attribute indicating the quality of the loan (*ok* or *bad*). Although there are only 682 individuals, the database contains a total of 1,079,680 records, mainly as the result of the 1,056,320 records in the **transaction** table. The database contains 76 bad loans (11.1%). We are looking for rules covering at least 170 individuals (25%).

Two sets of experiments were performed, one for each target value. Each set consisted of four runs, each with a different rule measure to be optimised. Of the six available measures, three tend to produce uninteresting results in the presence of numeric data, as they favour patterns with a high coverage. This results in an emphasis on non-selective refinements, which are easy to produce with

numeric attributes. Of the three measures coverage, support and sensitivity, only the latter has been included for demonstration.
Table 5.1 and Table 5.2 give an overview of the results obtained for the target values *bad* and *ok*, respectively. *Empty rule* refers to the score for the trivial rule → *true*. *Minimum score* refers to the user-defined threshold for rules to be reported. *Best* is the score of the best rule. *Nr. rules* refers to the number of rules reported. Of course, this number depends directly on the user-defined minimum score. The tables additionally indicate the number of data mining primitives executed, and the number of hypotheses these primitives represent. Finally, the running time in seconds is given.

	novelty	accuracy	sensitivity	specificity
empty rule	0.0	0.111	1.0	0.0
minimum score	0.02	0.111	0.9	0.7
best	0.0219	0.176	1.0	0.757
nr. rules	8	489	695	280
nr. primitives	1897	1776	2731	3781
nr. hypotheses	135,771	114,122	297,553	107,427
time	1844	1454	1226	514

Table 5.1 Results on Financial (target *bad*).

	novelty	accuracy	sensitivity	specificity
empty rule	0.0	0.889	1.0	0.0
minimum score	0.02	0.889	0.9	0.7
best	0.0375	0.989	1.0	0.96
nr. rules	164	511	80	451
nr. primitives	2043	2001	2731	1838
nr. hypotheses	173,604	113,236	298,484	75,956
time	918	786	1197	361

Table 5.2 Results on Financial (target *ok*).

These results demonstrate the large number of hypotheses that is considered during a typical run. Although in these experiments an average of over 164,000 hypotheses per run was tested, this was achieved by only 2349 queries (on average 70 hypotheses per query). This demonstrates the effectiveness of the data mining primitives employed. On average, each experiment lasted 1037 seconds, resulting in an average of 158 hypotheses being considered per second.
Most rules involve information from three or more tables (typically **account**, **order**, and **transaction**). The attribute conditions combine nominal and numeric tests, which most multi-relational rule discoverers do not allow. The best rules represent subgroups that show a significant change in distribution of the target value (e.g. (11.1%, 88.9%) vs. (17.6%, 82.4%), which corresponds to a p-value of 0.0009). This can be of great practical use in the financial domain in question, for example, to improve loan approval procedures.
Figure 5.4 shows two typical patterns (target *bad* and optimised for novelty) discovered. Further informal inspection of the rule sets produced shows that Safarii tends to report interesting subgroups in multiple copies. The rule sets may contain logically equivalent rules, for example by swapping conditions in the selection nodes. Furthermore, it will often report rules that only differ in details that seem to be irrelevant to the overall dependency discovered. Alternatively, it will produce variants that differ considerably in structure, but that nevertheless produce (almost) identical subgroups. The rules described in Figure 5.4 illustrate these effects.

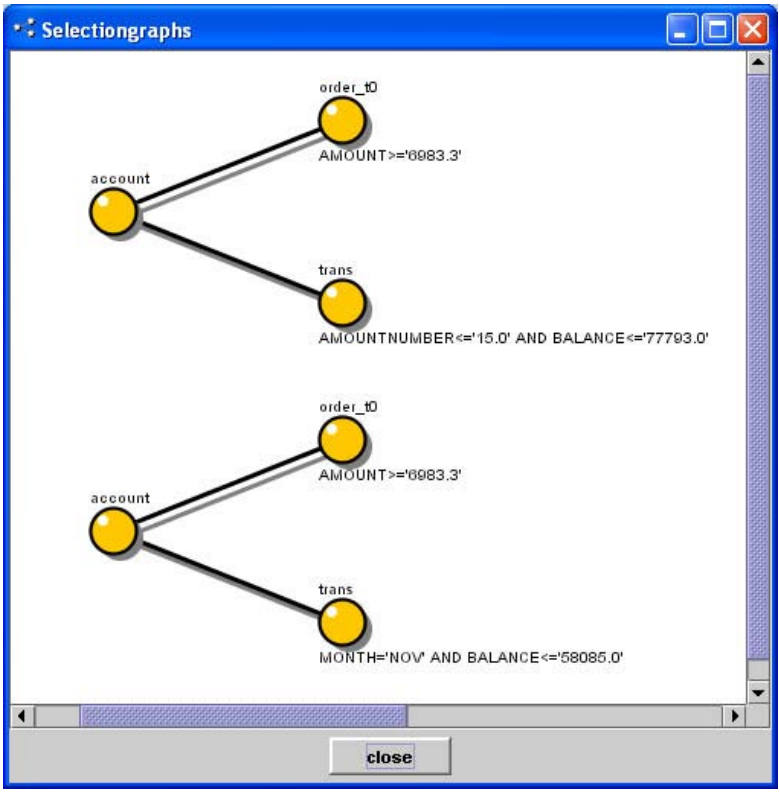


Figure 5.4 Two rules covering 183 and 175 loans (35 and 34 *bad*) respectively.

Thus far the rules discovered have been treated as an independent rule set, with the disadvantage of having many similar rules. Instead, we can consider the value of individual rules in the context of the remaining rules in the rule set. A good way to do this is by means of an ROC analysis (Receiver Operating Characteristics) [35, 39, 86]. Each rule will effectively be treated as a classifier, and its performance plotted in the two-dimensional space defined by its *true positive rate* (corresponds to the sensitivity of the rule) and the *false positive rate* (1 – specificity). The ROC analysis assumes that different classifiers may be optimal under different circumstances, depending on the misclassification cost (which may not be specified at the time of analysis). By considering the ROC plot, and the classifiers that are optimal for each potential specification of the misclassification cost, we can filter our rule set, and discard rules that can never be optimal. As classifiers on a straight line (known as *iso-performance* lines) have the same cost, the convex hull of the set of points in the ROC plot can be used to find the optimal rules. The convex hull includes the points (0, 0) and (1, 1), for the rules → *false* and → *true*, respectively.

Figure 5.5 shows the resulting ROC plot for one of the experiments (target *ok*, optimised for novelty) mentioned above. The diagonal line represents the minimum coverage of 25%. Clearly, only a fraction of the rules lie on the convex hull, even when considering the duplicate subgroups present. Novelty is a good score function in relation to ROC analysis, as it scores rules on the basis of equal costs. Rules with the same novelty appear on iso-performance lines of slope 1, and better

rules appear towards the top left [35]. It should be noted that although having a condensed rule set is attractive, the rules below the convex hull may still be important for more qualitative reasons.

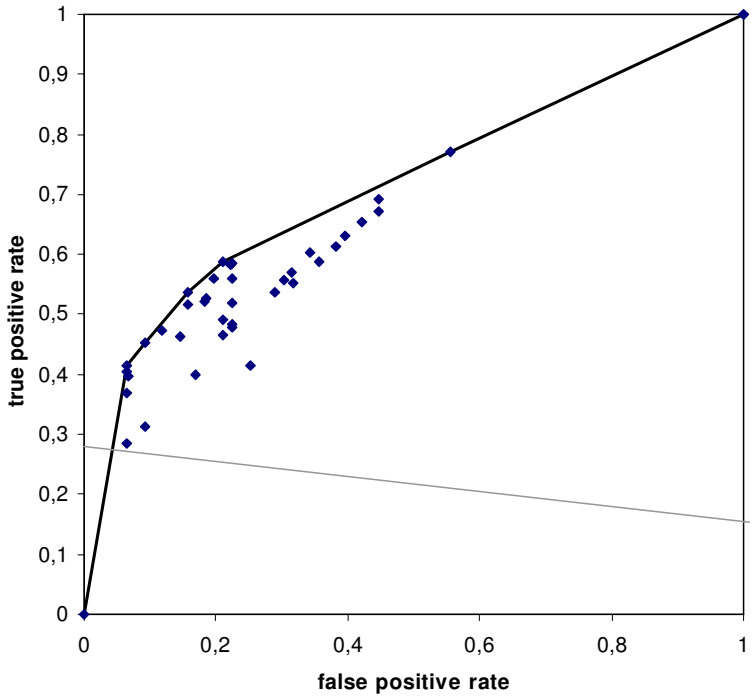


Figure 5.5 ROC plot of the rules discovered with target *ok* and optimised for novelty.

5.4 Related Work

The Safarii system is inspired by propositional subgroup miners such as Explora [51, 52, 53] and DataDistilleries [46], a commercial Data Mining system marketed by SPSS (formerly Data Surveyor by Data Distilleries B.V.). Like Safarii, these systems offer a range of analysis scenarios by setting parameters such as the search strategy and score function. Explora is limited in its ability to handle large quantities of data. DataDistilleries, however, handles data in the same way as Safarii. By using a number of (propositional) data mining primitives, it is able to deal with industrial-size databases.

Within the field of ILP, there are two major systems concerned with rule and subgroup discovery: MIDOS [31, 107] and Tertius [36]. Both systems use a top-down refinement operator that is non-redundant. Furthermore, these systems are able to use optimistic estimates (see Section 12.6) for a given part of the search space, which allows considerable pruning. Both systems require numeric data to be discretised, which is a serious limitation compared to Safarii. On the other hand, the manageable size of the search space that results from fixing numeric thresholds, combined with the optimal refinement operator, makes exhaustive search attainable in many problem domains. Neither system provides support for relational database systems.

A third ILP system, called WARMR [22] discovers rules in two steps. First, a collection of frequent patterns is produced. Then, each pattern is split into a head and a body of a rule in several ways. The resulting rule set contains rules with a variety of heads, unlike the results of MIDOS, Tertius, and Safarii, which work with a predetermined target. WARMR does not fully support numeric data, as the notion of frequent patterns conflicts with the freedom of selecting thresholds from a continuous domain.

This page intentionally left blank

6 Multi-Relational Decision Tree Induction

In the previous chapter, we have considered a method for discovering local models of the data. Each local model, a multi-relational rule, describes a dependency that holds for some part of the space of individuals, and all rules do so independently of one another. Particular individuals may be covered by multiple rules, or in fact by no rule at all. Such an approach may be interesting if a descriptive analysis of the database is required. However, if we require a predictive model of the domain, which makes a unique prediction for each individual and can deal with every conceivable individual, we need a global model. In this chapter, we will consider a popular class of global models, *decision trees*, and demonstrate how they can be induced from a multi-relational database.

Decision trees have been popular, especially in Propositional Data Mining, due to their good performance as well as because of the elegant and intuitive representation of the knowledge that is discovered [88]. Decision trees are typically induced by considering a number of tests, and picking one that divides the database into two or more mutually exclusive subgroups. This process is then repeated recursively to each of these branches, to grow a tree. In the propositional case, the test performed at an internal node of the tree can be a simple condition on one of the input attributes, which corresponds to one branch. The condition is then negated to form the second branch. Alternatively, a nominal attribute with cardinality n may be used to form n branches. If we now want to determine which subgroup a particular internal node or leaf represents, we simply take the conjunction of conditions along the path from the root to the node in question.

In multi-relational decision trees, the situation is more complex. As was outlined in Chapter 4, it is generally not possible in MRDM to produce two mutually exclusive subgroups by using complementary operators, or to divide a subgroup on the basis of nominal values. Instead a more elaborate way of producing complementary refinements of a selection graph must be devised. In this chapter, we will show how the existing collection of refinements can be extended into a collection of pairs of complementary refinements. Each pair can thus be used to split a group into

two subgroups. Consequently, the decision trees that we build will be binary trees. It turns out that the selection graph language is not expressive enough to accommodate for the splitting of subgroups. In the next section, we will introduce an extension of the regular selection graphs that is powerful enough.

Example 6.1 Consider a database of bank accounts and transactions. We may want to split the set of accounts that have a transaction into a set of accounts with at least one withdrawal, and the complement of that set. Clearly, this complement is not equal to the set of accounts with at least one deposit, as this does not exclude accounts with both withdrawals and deposits. Rather, the complement is equal to the set of accounts with a transaction, but none of the transactions is a withdrawal.

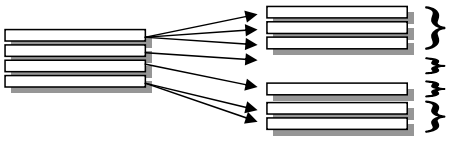
The induction of decision trees in first-order logic has been studied by several researchers [12, 14, 66, 105]. These approaches share a common Divide-and-Conquer strategy, but produce different flavours of decision trees. For example [66] discusses the induction of regression trees, whereas [14] discusses the induction of decision trees for clustering. In [12] an overview of potential uses of decision trees within a single framework is given. However, these papers have largely focused on induction-parameters such as the choice of splitting criterion or stopping criterion. None of these papers provide a good solution for the representation of patterns associated with the leaves and internal nodes of the decision tree. In this chapter, we will describe a general framework for the top-down induction of decision trees in the context of MRDM.

6.1 Extended Selection Graphs

In order to describe the multi-relational patterns associated with each node in a decision tree, we introduce the language ESG of *extended selection graphs*:

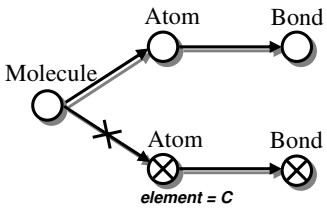
Definition 6.1 An *extended selection graph* G is a directed graph (N, E), where N is a set of triples (t, C, s) called *selection nodes*, t is a table in the data model and C is a, possibly empty, set of conditions on attributes in t of type $(t.a\ \theta\ c)$; θ one of the following operators, $=, \leq, \geq$. s is a flag with possible values *open* and *closed.*
E is a set of tuples (p, q, a, e) called *selection edges*, where p and q are selection nodes and a is an association between $p.t$ and $q.t$ in the data model. e is a flag with possible values *present* and *absent.*
The extended selection graph contains at least one node n_0 that corresponds to the target table t_0.

Extended selection graphs can be represented graphically as labelled directed graphs. The value of s is indicated by the absence or presence of a cross in the node, representing the value *open* and *closed*, respectively. The value of e is indicated by the absence or presence of a cross on the arrow, representing the value *present* and *absent* respectively.
Intuitively, extended selection graphs can be interpreted as follows. Every edge between p and q, together with the associated conditions $q.C$, imposes some constraints on how multiple records in table $q.t$ relate to a single record in table $p.t$. The association between $p.t$ and $q.t$ imposes some grouping on the records in $q.t$, and the combination of edge and conditions selects some of these groups in $q.t$, consequently selecting some records in $p.t$. Multiple edges coming from one node simply indicate multiple conditions on the records belonging to that node. Such a grouping is demonstrated below.



A *present* edge combined with a list of conditions selects those groups for which there is at least one record that respects the list of conditions. An *absent* edge combined with a list of conditions selects only those groups for which there is not a single record that satisfies the list of conditions. The selection associated with any subgraph is the combined result of all such individual constraints within the subgraph on groups of records. This means that any subgraph that is pointed to by an *absent* edge should be considered as a joint set of negative conditions. The flag *s* associated with nodes of the extended selection graph has no effect on the selection. Rather, it is used to indicate whether a particular node in an extended selection graph is a candidate for refinement. Only *open* nodes can be the subject of future refinements (see Section 6.1.1).

Example 6.1 The following extended selection graph selects those molecules that have at least one atom with a bond, but for whom none of these atoms are C-atoms:



The following algorithm shows how extended selection graph can be translated into SQL. The translation provides a formal semantics for selection graphs. The algorithm will produce a list of tables *table_list*, a list of join conditions *join_list*, and a list of conditions *condition_list*, and combine these to produce an SQL-statement. The algorithm effectively produces a join of all the tables associated with an *open* node. For each subgraph attached to an *absent* edge, a subquery is produced by calling the *TranslateSubgraph* procedure.

```
TranslateExtendedSelectionGraph(extended selection graph S)

    table_list = ''
    condition_list = ''
    join_list = ''
    for each node i in S.N
        if (i.s = ' open ')
            table_list.add(i.table_name + ' T' + i)
            for each condition c in i
                condition_list.add('T' + i + '.' + c)
    for each edge j in S.E
        if (j.e = 'present')
            if (j.right_node.s = ' open ')
                join_list.add(j.left_node + '.' + j.left_attribute + ' = ' +
```

```
                                        j.right_node + '.' + j.right_attribute)
            else
                    join_list.add(j.left_node + '.' + j.left_attribute + ' not in ' +
                                translate_subgraph(subgraph(S, j.right_node),
                                                    j.right_attribute))
    return 'SELECT DISTINCT T0.' + n0.primary_key +
            ' FROM ' + table_list + ' WHERE ' + join_list + ' AND ' + condition_list
```

TranslateSubgraph(extended selection graph *S*, key *K*)

```
    table_list := ''
    condition_list := ''
    join_list := ''
    for each node i in S.N
            table_list.add(i.table_name + ' T' + i)
            for each condition c in i
                    condition_list.add( 'T' + i + '.' + c)
    for each edge j in S.E
             join_list.add(j.left_node + '.' + j.left_attribute + ' = ' +
                            j.right_node + '.' + j.right_attribute)
    return 'SELECT T0.' + K +
            ' FROM ' + table_list + ' WHERE ' + join_list + ' AND ' + condition_list
```

Example 6.2 The algorithm *TranslateExtendedSelectionGraph* will produce the following SQL statement for the extended selection graph given in Example 6.1:

```
SELECT DISTINCT T0.id
FROM molecule T0, atom T1, bond T2
WHERE T0.id = T1.molecule_id AND T1.id = T2.atom1_id
AND T0.id not in
     (SELECT T3.molecule_id
      FROM atom T3, bond T4
      WHERE T3.id = T4.atom1_id AND T3.element = 'C')
```

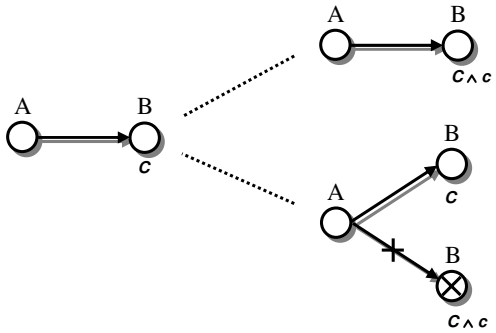
6.1.1 Refinements

As was described earlier, the selection graphs will be used to represent sets of objects belonging to nodes or leaves in a decision tree. Whenever a new split is introduced in the decision tree, the current selection graph is in fact being refined in two ways. The refinement operator ρ_{ESG}, defined below, of an extended selection graph *G* will be used as to generate potential splits in the multi-relational decision tree. The refinements are introduced in pairs of complimentary operations:

Condition refinement

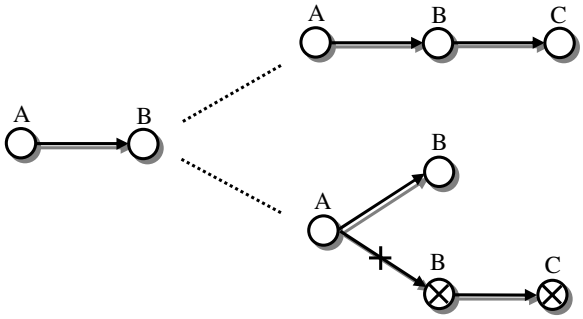
- the refinement adds a condition to an *open* selection node in *G* without actually changing the structure of *G*.
- in the case that the (*open*) node that is refined does not represent the target table, this refinement copies the subgraph that contains the node in question. This copy is then attached to the root-node by an *absent* edge. All new nodes are marked *closed*. Finally, the condition list of the copy of the node that is refined is extended by the new condition. If the node in

question does represent the target table, the condition is simply negated and added to the current list of conditions for this node.



Association refinement

- this refinement instantiates an association in the data model as a *present* edge together with its corresponding table. The resulting subgraph is then attached to an *open* node in *G*. The new node is marked *open*.
- in the case that the node that is refined does not represent the target table, this refinement copies the subgraph that contains the node in question. This copy is then attached to the root-node by an *absent* edge. An association in the data model is instantiated as a *present* edge together with its corresponding table, and attached to the copy of the node that is refined. If the node that is refined does represent the target table, the association is simply instantiated as an *absent* edge together with its corresponding table and added to *G*. All new nodes are marked *closed*.



In order to prove that ρ_{ESG} is a top-down refinement operator, we first prove the following:

Proposition 6.1 Let *G* be an extended selection graph and *n* a node in *G*. Let *S* be the set of records corresponding to *n*. Let *S′* be the set of records corresponding to *n* after addition to *n* of a condition, a *present* edge and node, or an *absent* edge and node. Then $S' \subseteq S$.

The proof follows directly from the semantics of conditions, *present* and *absent* edges. Any addition will act as a filter on the represented set of records.

Proposition 6.2 Let G be an extended selection graph and n a node in G. Let S and S' be the set of records corresponding to n before and after a refinement to a node m in the subgraph starting at n, respectively. Then $S' \subseteq S$.

Proof Inspection of ρ_{ESG} shows that all nodes in a subgraph that is attached by an *absent* edge are *closed*, and *open* otherwise. A path between two *open* nodes therefore always consists of *present* edges. n and m are both *open* and hence connected by a *present* path. Now consider two nodes n_1 and n_2 connected by a *present* edge, along with two sets of records at n_2, S_2 and S_2', such that $S_2' \subseteq S_2$. We can easily show that for the corresponding sets S_1 and S_1' at n_1, $S_1' \subseteq S_1$ holds. By induction we can now show the same property for any *present* path between two nodes. Because Proposition 6.1 applies to m, we get that $S' \subseteq S$.

□

By applying Proposition 6.2 to the root-node n_0 of an extended selection graph that is being refined, we can see that the refinement in the subgraph works as a filter on the individuals. The possible *absent* subgraph introduced by the two complementary refinements works as an additional filter. Hence:

Proposition 6.3 ρ_{ESG} is a top-down refinement operator.

6.2 Multi-Relational Decision Trees

Top-down induction of decision trees is basically a Divide-and-Conquer algorithm. The algorithm starts with a single node at the root of the tree, which represents all individuals. By analysing all possible refinements of the empty extended selection graph, and examining their quality by applying some interestingness measure, the optimal refinement is determined. This optimal refinement together with its complement is used to create the patterns associated with the left and the right branch, respectively. Based on the stopping criterion, it may turn out that the optimal refinement and its complement do not give cause for further splitting. In this case, a leaf node is introduced instead. Whenever the optimal refinement does provide a good split, a left and right branch are introduced and the procedure is applied to each of these recursively.

```
BuildTree(tree T, database D, pattern P)

    R = OptimalRefinement(P, D)
    if StoppingCriterion(R)
        T = leaf(P)
    else
        P_left = R(P)
        P_right = R_compl(P)
        BuildTree(left, D, P_left)
        BuildTree(right, D, P_right)
        T = node(left, right, R)
```

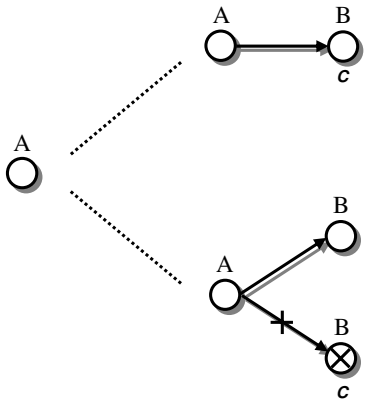
The function *OptimalRefinement* takes the current pattern P and considers every possible generic positive refinement with respect to the data model of D. For each of these generic refinements a multi-relational data mining primitive call is sent to the server that handles the data, and the necessary statistics are retrieved. The statistics describe the support of all possible refinements

derived from the current generic refinement. However, these counts also imply the support of the complementary negative refinements. This is explained in more detail in Section 10.4. The two sets of counts are used to compute the interestingness measure of choice. This process is repeated for every possible generic positive refinement and the optimal refinement is returned.
The function *StoppingCriterion* determines whether the optimal refinement leads to a good split based on the statistics associated with the optimal refinement and its complement. A range of stopping criteria can be used, depending on the induction paradigm (classification, clustering, regression, etc.), but the actual choice is immaterial to the present discussion.

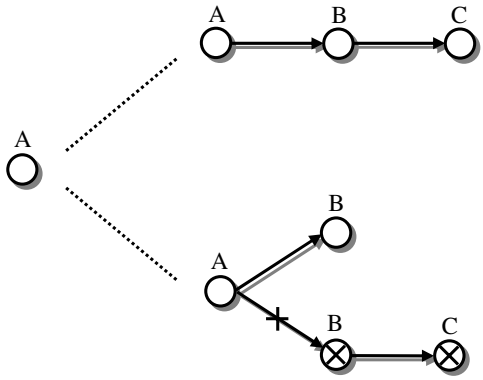
6.3 Look-Ahead

As is common in Data Mining, the generic algorithm for multi-relational decision tree induction presented here is greedy. At each node, it will consider a list of possible splits, and choose the most promising. It will never backtrack on this decision. In general, this greedy approach tends to work well. In a multi-relational setting however, there are often cases where a seemingly irrelevant refinement is necessary to facilitate further refinements. The problem occurs when a refinement introduces a new table in the extended selection graph. If the multiplicity of the association is 1..n or 1 (rather than 0..n or 0..1), then the association refinement will not cover fewer individuals and thus the complement will be empty. As a result, the split will be useless, and will most likely not be selected as the optimal split. Yet, this refinement is essential in including a whole range of new features stemming from the new table.
The solution is to use a form of look-ahead [12]. Whenever an association refinement of the described type is considered, it is immediately followed by a further refinement on this new table. For a given node in the tree, we will thus consider all the regular splits, plus the splits of the look-ahead, and choose the optimal as usual. Instead of thinking of this look-ahead as two consecutive refinements, we will introduce two new pairs of complementary refinements with the desired effect. This avoids the initial irrelevant split and the resulting empty leaf in the tree. The pairs correspond to an association refinement followed by a condition refinement, and a second association refinement, respectively.

Look-ahead condition refinement



Look-ahead association refinement



6.4 Two Instances

Decision tree induction algorithms come in many flavours. They can be characterised by how they implement a small number of parameters, including notably the splitting heuristic (e.g. information gain, Gini index, intra-cluster variance), the stopping criterion and the pruning mechanism [12]. The multi-relational decision tree framework presented in the previous sections is intended to be general, and therefore does not fix any of these parameters. If we want to obtain empirical results however, we require an instance of the presented generic algorithm that does make clear choices for the parameters mentioned. In the remainder of this chapter, we describe two implementations by two different research groups, that build on the tree induction machinery defined previously. The two approaches build trees for classification and regression, respectively.

6.4.1 MRDTL

The MRDTL (Multi-Relational Decision Tree Learning) [74] system was developed by H. Leiva, A. Atramentov and V. Honavar from Iowa State University. It implements multi-relational decision trees with the aim of building classifiers, in a way similar to that of C5.0 [88]. As such, it works similarly to Tilde [12], but in a database rather than an ILP-setting. MRDTL uses the classical information gain heuristic in order to find the optimal split. It does not currently incorporate any pruning mechanism.

Leiva et al. present a number of experiments using MRDTL. We will focus on one experiment involving the Mutagenesis database mentioned earlier. This dataset also features in a number of experiments involving different systems that will be presented in following chapters.

The Mutagenesis database describes molecules falling into two classes, *mutagenic* and *non-mutagenic* [96]. The dataset analysed is known as the 'regression-friendly' dataset, and consists of 188 molecules. The database consists of 26 tables, of which three tables directly describe the graphical structure of the molecule (**molecule**, **atom** and **bond**). The remaining 23 tables describe the occurrence of predefined functional groups, such as benzene rings. Four different experiments were performed, using different settings, or so-called *backgrounds*. They will be referred to as experiments B1 to B4:

- B1: the atoms in the molecule are given, as well as the bonds between them; the type of each bond is given as well as the element and type of each atom.
- B2: as B1, but continuous values about the charge of atoms are added.
- B3: as B2, but two continuous values describing each molecule are added.

- B4: as B3, but knowledge about functional groups is added.

	Progol	FOIL	Tilde	MRDTL
B1	76%	61%	75%	**67%**
B2	81%	61%	79%	**87%**
B3	83%	83%	85%	**88%**
B4	88%	82%	86%	**-**

Tabel 6.1 Results on Mutagenesis.

Leiva et al. describe results obtained in comparison with the three ILP classifiers Progol, FOIL and Tilde [9]. Except for B1, where an empty tree was induced, MRDTL appears to outperform the competing algorithms. Especially the comparison with Tilde is interesting because of the similarities in choice of splitting heuristic and refinements. The increased performance is most likely due to the way Tilde deals with numeric attributes. It effectively globally discretises them by only considering a small number of thresholds. In contrast, our tree induction framework, and specifically MRDTL will consider all appropriate numeric thresholds. As a consequence, MRDTL runs somewhat slower than Tilde, although notably faster than Progol and FOIL.

6.4.2 Mr-SMOTI

A more substantial extension of the presented framework for induction of decision trees from multi-relational data is achieved by the Mr-SMOTI system, developed by A. Appice, M. Ceci and D. Malerba from Bari University [7]. It aims at building regression trees, hierarchical structures that can be used to approximate a continuous attribute of a given individual, as opposed to the nominal attributes that are the target of classification trees (e.g. MRDTL). Mr-SMOTI is the multi-relational counterpart of SMOTI, a regression tree induction algorithm that works exclusively on propositional data.

As our framework suggests, the Mr-SMOTI system produces a hierarchical structure of mutually exclusive subgraphs (by means of extended selection graphs) that exhibit a minimal variance in the target attribute. However, it adds to that linear models associated with nodes and leaves that make the approximation more precise. At each point in the construction of the tree, it considers the usual split-refinements, as well as a regression-refinement, which will add a, to this point unused, continuous attribute to the composition of linear models produced along the path to the current node. As such, it produces a composite linear model that is progressively more local towards the leaves of the tree. To accommodate this functionality, the ESG language has been extended with administration of linear models at each selection node.

Appice et al. [7] describe an experimental comparison with Tilde-RT [9], a regression tree instance of the well-known ILP-system. Experiments were performed on two different datasets, a regression-variant of the Mutagenesis database, and Biodegradability [29]. On these two datasets, Mr-SMOTI shows a slightly, but consistently better performance compared to Tilde-RT.

This page intentionally left blank

7 Aggregate Functions

As was outlined in Chapter 4, we can identify two important approaches to capturing local structure. The first approach describes groups of records by means of existential features. The second approach focuses on features of the group as a whole, by means of aggregate functions. The MRDM technology presented so far has been based on capturing local structure exclusively through existential features. In this and the following two chapters, we investigate the extra benefits that aggregate functions have to offer. The present chapter describes the basic properties of aggregate functions, and how a good set of features can be selected from the potentially infinite set of aggregate functions. Having a set of well-chosen aggregate functions will allow us to describe the essence of the structural information over a wide variety of structures.

7.1 Aggregate Functions

We define an aggregate function as a function that takes as input a set of records in a database, related through the associations in the data model, and produces a single value as output. Aggregate functions will be used to project information stored in several tables on one of these tables, essentially adding virtual attributes to this table. In the case where the information is projected on the target table, and structural information belonging to an individual is summarised as a new feature of that individual, aggregate functions can be thought of as a form of feature construction.
The broad definition used here includes aggregate functions of a great variety of complexity. An important aspect of the complexity of an aggregate function is the number of (associations between) tables it involves. As each aggregate function essentially considers a subset of the data model, we can use our three previously defined complexity-measures for data models to characterise aggregate functions. Specifically association-depth is useful to classify aggregate functions. An aggregate function of association-depth 0 involves just one table, and is hence, a case of propositional feature construction. In their basic usage, aggregate functions found in SQL (count, min, sum, etc.) have an association-depth of 1, whereas association-deeper aggregate functions represent some form of

multi-relational pattern (benzene-rings in molecules, etc.). Using this classification of association-depth, we give some examples to illustrate the range of possibilities.

$d_a(A) = 0$:

- Propositions (adult = (age $\geq$ 18))
- Arithmetic functions (area = (width $\cdot$ length))

$d_a(A) = 1$:

- Count, count with condition
- Count distinct
- Min, max, sum, avg
- Exists, exists with condition
- Select record (eldest son, first contract)
- Mode

$d_a(A) > 1$:

- Exists substructure
- Count substructure
- Conjunction of aggregate functions (maximum count of children)

Clearly the list of possible classes of aggregate functions is long, and the number of instances is infinite. An overview of interesting classes of aggregate functions, along with a hierarchy of increasing complexity is given in [85]. In order to arrive at a practical and manageable set of aggregate functions, we will have to drastically limit the range of classes and instances. Apart from deterministic and heuristic rules to select good candidates, pragmatic limitations to a small set of aggregate function classes are unavoidable. In this thesis, we have chosen to restrict ourselves to the classes available in SQL and combinations thereof.

7.2 Aggregation

Let us consider two tables *P* and *Q*, neither of which needs to be the target table, that are joined by an association *A*. By aggregating over *A*, information can be added to *P* about the structural properties of *A*, as well as the data within *Q*. To aggregate *Q*, a set of aggregate functions of association-depth 1 is needed.
As was demonstrated before, the multiplicity of association *A* influences the search space of multi-relational patterns involving *A*. The same is true for aggregation over *A* using aggregate functions. The choice of aggregate functions depends on the multiplicity of *A*. In particular, if we aggregate *Q* over *A,* only the multiplicity on the side of *Q* is relevant. This is because an association generally describes two relationships between the records in both tables, one for each direction. The following four options exist:

1 For every record in *P* there is but a single record in *Q*. This is basically a look-up over a foreign key relation, and no aggregate functions are required. A simple join will add all non-key attributes of *Q* to *P*.

0..1 Similar to the 1 case, but now a look-up may fail because a record in *P* may not have a corresponding record in *Q*. An outer join is necessary, which fills in NULL values for missing records.

1..n For every record in *P*, there is at least one record in *Q*. Aggregate functions are required in order to capture the information in the group of records belonging to a single record in *P*.

0..n Similar to the 1..n case, but now the value of certain aggregate functions may be undefined due to empty groups. Special care will need to be taken to deal with the resulting NULL values.

Let us now consider the 1..n case in more detail. *A* imposes a grouping on the records in *Q*. For *m* records in *P* there will be *m* groups of records in *Q*. Because of the set-semantics of relational databases every group can be described by a collection of histograms or data-cubes. We can now view an aggregate function instance as a function of one of these types of histograms. For example the *mode* aggregate function for an attribute *Q.a* simply returns the value corresponding to the highest count in the histogram of *Q.a*. Note that *m* groups will produce *m* histograms and consequently *m* values for one aggregate function instance, one for each record in *P*. The notion of functions of histograms helps us to define relevant aggregate function classes.

count The *count* aggregate function is the most obvious aggregate function through its direct relation to histograms. The most basic instance without conditions simply returns the single value in the 0-dimensional histogram. Adding a single condition requires a 1-dimensional histogram of the attribute involved in the condition. For example, the number of sons in a family can be computed from a histogram of gender of that family. An attribute with a cardinality *c* will produce *c* aggregate function instances of *count* with one condition.

There is some overlap in the patterns that can be expressed using the *count* aggregate function and those expressed by existential features. Testing for a count greater than zero obviously corresponds to existence. Testing for a count greater than some threshold *t* however, requires a refinement-depth of $O(t^2)$ in existential features, as a quadratic number of inequalities needs to be added in order to guarantee correct counting. Things become even worse for existential features when the < operator is used, as it requires the use of negation in a way that (extended) selection graphs (as well as the language bias of many ILP algorithms) does not cater to. The use of the *count* aggregate function is clearly more powerful in these respects.

min and max The two obvious aggregate functions for numeric attributes, *min* and *max*, exhibit similar behaviour. Again there is a trivial way of computing *min* and *max* from the histogram; the smallest and largest values having a non-zero count, respectively. The *min* and *max* aggregate functions support another type of constraint commonly used in existential feature-based algorithms, *exists* with a numeric constraint. Lemma 4.1 describes the correspondence between the minimum and maximum of a group of numbers, and the occurrence of particular values in the group. It simply states that testing whether the maximum is greater than some threshold is equivalent to testing whether any value is greater than *t*. Analogous for *min*. It is important to note the combination of *max* and ≥, and *min* and ≤. If *max* were to be used in combination with ≤ or =, then the existential feature equivalent would again require the use of negation.

Such use of the *min* and *max* aggregate function gives us a natural means of introducing the universal quantor ∀: all values are required to be above the minimum, or below the maximum. Another advantage of the *min* and *max* aggregate function is that they each replace a set of binary *exists* aggregate function instances (one for each threshold), making the representation of virtual attributes a lot more compact. The following lemma illustrates how the *min* and *max* function can easily capture local structure in a way that would require undesirable universal quantification (or negation) otherwise.

Lemma 7.1 Let B be a bag of real numbers, and t some real, then

$$\max(B) = t \textit{ iff } \forall v \in B: v \leq t \wedge \exists v \in B: v = t,$$
$$\min(B) = t \textit{ iff } \forall v \in B: v \geq t \wedge \exists v \in B: v = t,$$
$$\max(B) \leq t \textit{ iff } \forall v \in B: v \leq t,$$
$$\min(B) \geq t \textit{ iff } \forall v \in B: v \geq t.$$

In short, we can conclude that on the level of aggregation ($d_a = 1$), aggregate functions can express many of the concepts expressible in existential features. They can even express concepts that are hard or impossible to express existentially.

Example 7.1 Consider the main three tables in the Mutagenesis database (see also Figure 3.3). We can enhance the molecule-specific information in **molecule** by aggregating the **atom** and **bond** table. The result is shown in Figure 7.1. The new table (or view) contains the original attributes, as well as aggregate functions of the attributes that appear in **atom** and **bond** as the result of the two one-to-many associations involved. There are two virtual attributes with *count* of association-width 0, and seven attributes of association-width 1. The aggregate functions applied depend on the type of the attribute in question.

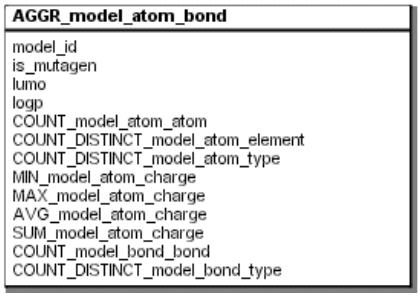


Figure 7.1 Results of aggregation on **molecule** in Mutagenesis.

7.3 Aggregate Functions & Association-width

If we include the *exists* function (or *count*), we can think of an aggregate function-based approach as a generalisation of the existential methods that are so common in Structured Data Mining. An elegant feature of most methods based on existential features is that after introducing a new table, not simply the occurrence of a record within the group is considered, but also the occurrence of records that satisfy certain conditions on the value of its attributes. Such patterns are typically considered by progressive refinements on the initial pattern, as an integral part of the regular search for patterns. We can do the same with aggregate functions by not only considering functions on the whole group, but also on subsets that are identified by conditions on attributes. We can thus characterise groups in a variety of ways, even if we only have a small collection of aggregate function classes.

Clearly, the number of aggregate function instances explodes with the higher association-width obtained in this way. We end up with a number of features that is impractical in situations where we want to use aggregation as a means of pre-processing the data prior to the actual Data Mining [72]. This potential combinatorial explosion will be dealt with in two alternative ways:

- set a low upper bound on the association-width. This is the necessary choice if a selection of features needs to be fixed during pre-processing. In Chapter 8, we introduce a method that

repeatedly flattens pairs of tables by aggregation to arrive at a single target table containing a range of multi-relational features, a process known as *propositionalisation*. For this method we restrict ourselves to the aggregate function classes available in SQL, and fix $w_a \leq 1$: for *count* we add up to one condition on a nominal attribute (no numeric conditions), and for the remaining classes no conditions as they already have an association-width of 1.

- allow any association-width, but only through progressive refinement. This approach only works if patterns with aggregate functions are considered dynamically, rather than prior to Data Mining. In Chapter 9, we examine an upgrade of the rule discovery algorithm presented in Chapter 5 that works along these lines.

If we want to consider aggregate functions of higher association-width by progressively refining the set of records that is aggregated, we will need to examine how the value of the aggregate functions depends on different sets of records. The following observations will be relevant for selecting proper refinements (see Section 9.3).

Definition 7.1

1. An aggregate function f is *ascending iff*, given sets of records S and S', $S' \subseteq S \Rightarrow f(S') \leq f(S)$.
2. An aggregate function f is *descending iff*, given sets of records S and S', $S' \subseteq S \Rightarrow f(S') \geq f(S)$.

Lemma 7.2

1. Aggregate functions *count*, *count distinct* and *max* are ascending.
2. Aggregate function *min* is descending.
3. Aggregate functions *sum* and *avg* are neither ascending nor descending.

Example 7.2 In the Mutagenesis problem database [95], there are tables for the concepts **molecule**, **atom** and **bond**. Some promising aggregate functions to describe the relation between **molecule** and **atom** are:

```
count(atom), min(atom.charge), avg(atom.charge)
```

We might also have similar functions for C-atoms in a molecule, or for C-atoms involved in a double bond. Clearly, the count decreases if we only consider a subset of atoms, such as the C-atoms. In contrast, the minimum charge will increase.

This page intentionally left blank

8 Aggregate Functions & Propositionalisation

In this chapter we will consider an approach to analysing multi-relational data that is different from the remainder of MRDM algorithms presented in this thesis. Rather than searching through a search space of multi-relational patterns and progressively discovering more complex structures, we will be using traditional propositional techniques. In order to apply these techniques with at least some degree of success, it will be necessary to transfer the original multi-table database into a single table, with one record for each individual. In order not to ignore the information in the tables other than the target table, the transformed table will contain an (often large) collection of features that capture different aspects of the structure in the individuals. This process of transforming a multi-relational database into a single table is known as *propositionalisation*.

The idea of propositionalisation (the construction of one table) is not new. Several relatively successful algorithms have been proposed in the context of Inductive Logic Programming (ILP) [6, 24, 65, 67, 97]. A common aspect of these algorithms is that the derived table consists solely of binary features, each corresponding to a (promising) clause discovered by an ILP-algorithm. This chapter however presents an alternative approach, based on the aggregate function machinery introduced in Chapter 7. It does not involve searching for relevant substructure, but rather relies on repeatedly summarising pairs of tables into a single new table by means of aggregate functions.

It may seem that having to select a fixed set of features before actually mining the data gives our approach a disadvantage compared to the more dynamic nature of regular MRDM algorithms. We will show, among others with experiments, that the propositionalisation approach has alternative benefits that allow it to compete with these algorithms. Their good performance can not only be attributed to how the transformation interacts with subsequent mining, but to a greater extent to the rich features that can be extracted from individuals by using aggregate functions.

8.1 Propositionalisation

In this section we describe the basic concepts involved in propositionalisation, and provide some definitions. We define propositionalisation as the process of transforming a multi-relational dataset into a propositional dataset with derived attribute-value features, describing specific structural properties of the individuals. The process can thus be thought of as summarising data stored in multiple tables in a single table (the *target table*) containing one record per individual. The aim of this process, of course, is to pre-process multi-relational data for subsequent analysis by attribute-value learners.

We will be using this definition in the broadest sense. We will make no assumptions about the data type of the derived attribute (binary, nominal, numeric, etc.) nor do we specify what language will be used to specify the propositional features.

Traditionally, propositionalisation has been approached from an ILP standpoint with only binary features, expressed in first-order logic [6, 24, 67, 65]. To our knowledge, the use of other aggregate functions than existence has been limited. One example is given in [29], which describes a propositionalisation-step where numeric attributes were defined for counts of different substructures. Friedman et al. [38] also mention aggregate functions as a means of establishing probabilistic relationships between objects in pairs of tables. It is our aim to analyse the applicability of a broader range of aggregate functions.

With a growing availability of algorithms from the fields of ILP and MRDM, one might wonder why such a cumbersome pre-processing step is desirable in the first place, instead of applying one of these algorithms to the multi-relational data directly. The following is a (possibly incomplete) list of reasons:

- Pragmatic choice for specific propositional techniques. People may wish to apply their favourite attribute-value learner, or only have access to commercial of-the-shelf Data Mining tools. Good examples can be found in the contributions to the Financial dataset challenge at PKDD conferences [106].
- Superiority of propositional algorithms with respect to certain Machine Learning parameters. Although extra facilities are quickly being added to existing Structured Data Mining engines, propositional algorithms still have a head start concerning handling of numeric values, regression, distance measures, cumulativity etc.
- Greater speed of propositional algorithms. This advantage of course only holds if the preceding work for propositionalisation was limited, or performed only once and then reused during multiple attribute-value learning sessions.
- Advantages related to multiple consecutive learning steps. Because two learning steps are applied, we are effectively combining two search strategies. The first step essentially transforms a multi-relational search space into a propositional one. The second step then uses these complex patterns to search deeper than either step could achieve when applied in isolation. This issue is investigated in more detail in the remainder of this section.

The term propositionalisation leads to some confusion because, although it pertains to the initial step of flattening a multi-relational database, it is often used to indicate the whole approach, including the subsequent propositional learning step. Because we are mostly interested in the two steps in unison, and for the sake of discussion, we introduce the following generic algorithm. The name is taken from Czech, and describes a two-step dance.

Polka (database D, data model M, integer r, p)

P = MRDM (D, M, r)
R = PDM (P, p)

The algorithm takes a database D and data model M (acting as declarative bias), and first applies a Multi-Relational Data Mining algorithm MRDM. The resulting propositional features P are then fed to a propositional Data Mining algorithm PDM, producing result R. We use the integers r and p very informally to identify the extent of the multi-relational and propositional search, respectively. Note that the propositionalisation step is independent of subsequent use in propositional learning.
In order to characterise the extent of the search more formally, we consider the complexity measures for the most complex patterns in the search space. We can thus characterise both individual patterns, as well as algorithms.

Lemma 8.1

1. $d_a(\text{Polka}) = d_a(\text{MRDM})$
2. $d_r(\text{Polka}) = d_r(\text{MRDM}) \cdot d_r(\text{PDM})$
3. $w_a(\text{Polka}) = w_a(\text{MRDM})$

Not surprisingly, the complexity of Polka depends largely on the complexity of the actual propositionalisation step. However, Lemma 8.1 2. demonstrates that Polka considers very refinement-deep patterns, in fact deeper than a multi-relational algorithm would consider in isolation. This is due to the combining of search spaces mentioned earlier. Later on we will examine the search restrictions that the use of aggregate functions have on the propositionalisation step and thus on Polka.

8.2 The RollUp Algorithm

With the foundations provided in the previous sections we can now define a basic propositionalisation algorithm. The algorithm will traverse the data model graph and repeatedly use the aggregation operation to project data from one table onto another, until all information has been aggregated at the target table. Although this repeated summarisation can be done in several ways, we will describe a basic algorithm, called RollUp.
The RollUp algorithm performs a depth-first search (DFS) through the data model, up to a specified depth. Whenever the recursive algorithm reaches its maximum depth or a leaf in the graph, it will "roll up" the relevant table by aggregating it onto the parent in the DFS tree. Internal nodes in the tree will again be aggregated after all its children have been aggregated. This means that attributes considered deep in the tree may be aggregated multiple times. The process continues until all tables are summarised onto the target table. Combined with a propositional learner we obtain an instance of Polka. The following pseudo code describes RollUp more formally:

```
RollUp (table T, data model M, integer d)

    V = T
    if d > 0
        for all associations A from T in M
            W = RollUp(T.getTable(A), M, d-1)
            S = Aggregate(W, A)
            V.add(S)
    return V
```

The effect of RollUp is that each attribute appearing in a table other than the target table will appear several times in aggregated form in the resulting view. This multiple occurrence happens for two

reasons. The first reason is that tables may occur multiple times in the DFS tree because they can be reached through multiple paths in the data model. Each path will produce a different aggregation of the available attributes. The second reason is related to the choices of aggregate function class at each aggregation along a path in the data model. This choice, and the fact that aggregate functions may be combined in longer paths, produces multiple occurrences of an attribute per path.
The association-depth of the deepest feature is equal to the parameter *d*. Each feature corresponds to at most one attribute aggregated along a path of depth d_a. The refinement-depth is therefore a linear function of the association-depth. As each feature involves at most one attribute, and is aggregated along a path with no branches, the association-width will always be either 0 or 1. This produces the following characteristics for RollUp. Use Lemma 8.1 to characterise Polka instantiated with RollUp.

Lemma 8.2

1. $d_a(\text{RollUp}) = d$
2. $d_r(\text{RollUp}) = d_a(\text{RollUp}) + 1$
3. $w_a(\text{RollUp}) = 1$

8.3 Experiments

In order to acquire empirical knowledge about the effectiveness of our approach, we have tested RollUp on three well-known multi-relational datasets. These datasets were chosen because they show a variety of data models that occur frequently in many multi-relational problems. They are Musk [27], Mutagenesis [95], and Financial [106].
Each dataset was loaded into the RDBMS Oracle. The data was modelled in MRML. Based on this declarative bias, the RollUp module produced one database view for each dataset, containing the propositionalised data. This was then taken as input for the common Machine Learning procedure C5.0. For quantitative comparison with other techniques, we have computed the average accuracy by leave-one-out cross-validation for Musk and Mutagenesis, and by 10-fold cross-validation for Financial.

8.3.1 Musk

The Musk database [27] describes molecules occurring in different conformations. Each molecule is either *musk* or *non-musk*, and one of the conformations determines this property. Such a problem is known as a multiple-instance problem, and will be modelled by two tables, **molecule** and **conformation**, joined by a one-to-many association. **Conformation** contains a molecule identifier plus 166 continuous features. **Molecule** just contains the identifier and the class. We have analysed two datasets, MuskSmall, containing 92 molecules and 476 conformations, and MuskLarge, containing 102 molecules and 6598 conformations. The resulting table contains a total of 674 features.
Table 1 shows the results of RollUp compared to other, previously published results [9, 27]. The performance of RollUp is comparable to Tilde, but below that of special-purpose algorithms.

Algorithm	MuskSmall	MuskLarge
Iterated-discrim APR	92.4%	89.2%
GFS elim kde APR	91.3%	80.4%
RollUp	**89.1%**	**77.5%**
Tilde	87.0%	79.4%
Back-propagation	75.0%	67.7%
C4.5	68.5%	58.8%

Table 8.1 Results on Musk.

8.3.2 Mutagenesis

Similar to the Musk database, the Mutagenesis database describes molecules falling in two classes, *mutagenic* and *non-mutagenic* (see Section 6.4.1). However, this time structural information about the atoms and bonds that make up the compound are provided. As before, we have analysed the four backgrounds B1 to B4 of the 'regression-friendly' dataset, consisting of 188 molecules.

Table 8.2 shows the results of RollUp compared to other, previously published results. Progol, FOIL and Tilde are ILP systems that produce classifiers based on rules or trees. MRDTL produces trees in a multi-relational setting. RELAGGS [68, 69] is a system that propositionalises a dataset by means of aggregate functions, not unlike RollUp (see Section 8.4.3).

Clearly RollUp outperforms the classic ILP methods on all backgrounds, except B4. Most surprisingly, RollUp already performs well on B1, whereas the ILP methods seem to benefit from the propositional information provided in B3. RollUp also is able to compete with MRDTL. The results of RollUp and RELAGGS are comparable.

	Progol	FOIL	Tilde	MRDTL	RELAGGS	RollUp
B1	76%	61%	75%	67%	86.7%	**86%**
B2	81%	61%	79%	87%	87.8%	**85%**
B3	83%	83%	85%	88%	86.7%	**89%**
B4	88%	82%	86%	-	88.8%	**84%**

Table 8.2 Results on Mutagenesis.

Example 8.1 The following tree of the B3 experiment illustrates the use of aggregate functions for structural descriptions.

```
CNT_BOND =< 26
    MODE_TYPE_ATOM [21 27] -> F
    MODE_TYPE_ATOM 22 -> F
    MODE_TYPE_ATOM 3
        MAX_CHARGE_ATOM =< 0.0
            MODE_TYPE_BOND 7 -> F
            MODE_TYPE_BOND 1 -> T
        MAX_CHARGE_ATOM  > 0.0 -> F
CNT_BOND  > 26
    LUMO =< -1.102
        LOGP =< 6.26 -> T
        LOGP  > 6.26 -> F
    LUMO  > -1.102 -> F
```

8.3.3 Financial

The third experiment covers the Financial database as described in Section 5.3. A near perfect score of 99.9% was achieved on this dataset. Due to the great variety of problem definitions described in the literature, quantitative comparisons with previous results are impossible. Similar (descriptive) analyses of loan-quality however, have never produced the pattern responsible for RollUp's good performance. The aggregation approach proved particularly successful on the large **transaction** table (1056320 records). This table has sometimes been left out of other experiments due to scalability problems.

8.4 Discussion

The experimental results in the previous section demonstrate that our approach is at least competitive with existing techniques based on existential features, such as Progol, Tilde, and MRDTL. Our approach has two major differences with these techniques, which may be the source of the good performance: the use of aggregate functions and the use of propositionalisation. Let us consider the contribution of each of these in turn.

8.4.1 Aggregate functions

There is an essential difference in the way a group of records is characterised by existential features and by aggregate functions. Existential features are based on the occurrence of one or more records in the group with certain properties. Aggregate functions on the other hand typically describe the group as a whole: each record has some influence on the value of the aggregate function. The result of this difference is that existential features and aggregate functions provide two unique feature-spaces to the learning procedure. Each feature-space has its advantages and disadvantages, and may be more or less suitable for the problem at hand.

Although the feature-spaces produced by existential features and aggregate functions have entirely different characteristics, there is still some overlap. As was shown in Chapter 7, some aggregate functions are very similar in behaviour to existential features. The common features in the two spaces typically

- select one or a few records in a group (*min* and <, *max* and >, *count* > 0 for some condition).
- involve a single attribute: $w_a \leq 1$
- have a relatively low association-depth.

If these properties hold, aggregate-based learning procedures will generally perform better, as they can dispose of the common selective aggregate functions, as well as the complete aggregate functions such as *sum* and *avg*.

Data models with a low association-depth are quite common in database design, and are called star-shaped ($d_a = 1$) or snowflake schemata ($d_a > 1$). The Musk dataset is the most simple example of a star-shaped model. The data model of Mutagenesis consists for the large part of a star-shaped model, and Financial is essentially a snowflake schema. Many real-world datasets described in the literature as SDM applications essentially have such a manageable structure. Moreover, results on these datasets frequently exhibit the extra condition of $w_a \leq 1$. Some illustrative examples are given in [29, 95].

8.4.2 Propositionalisation

According to lemma 8.2, Polka has the ability to discovery patterns that have a bigger refinement-depth than either of its steps has. This is demonstrated by the experiments with the particular instance of Polka presented here. RollUp produces relatively association-deep, and thus, refinement-deep features. These refinement-deep features are combined in the decision tree. Some

leaves represent very refinement-deep patterns, even though their support is still sufficient. This is an advantage of Polka (propositionalisation + propositional learning) over SDM algorithms in general.
Next to advantages related to expressiveness, there are more practical reasons for using Polka. Once the propositionalisation stage is finished, a large part of the computationally expensive work is done, and the derived view can be analysed multiple times. This not only provides a greater efficiency, but also gives the analyst more flexibility in choosing the right modelling technique from a large range of well-developed commercially available set of tools. The analyst can vary the style of analysis (trees, rules, neural, instance-based) as well as the Data Mining task (classification, regression).

8.4.3 Related Work

Two alternative MRDM approaches implement propositionalisation by means of aggregate functions. The RELAGGS system [68, 69] uses a slightly larger collection of aggregate function classes to summarise groups of records. The essential difference with RollUp however, is where aggregation is applied. Rather than recursively joining and aggregating pairs of tables towards the target table, RELAGGS propagates the key of the individual to each table, effectively turning the data model into a star schema (association-depth 1). It then applies aggregate functions to all tables, as if directly connected to the target table. This process produces the same result (given the same collection of aggregate function classes) whenever a database of association-depth 1 or less is processed. On association-deeper data models however, RollUp will nest aggregate functions (e.g. average of the count), whereas RELAGGS will aggregate over the transitive association.
Perlich et al. [85] describe another approach that treats the data model as a star schema, although not using propagated keys, but aggregate functions over the joins along paths in the data model starting from the target table. This work focuses on concept classes of aggregate functions that go beyond the basic functions employed by RollUp and RELAGGS. Experimental results show the clear benefit of using more elaborate aggregate functions. To some degree, these more complex functions correspond to the resulting nested functions of RollUp, or constructs described in Chapter 9.
A comparison between RELAGGS and two propositionalisation approaches based on binary features produced by ILP algorithms is presented in [69]. The results are not conclusive in favour of either approach, but aggregate functions appear to work better in domains where counting and dealing with numbers is relevant. Binary features are preferred in domains were the occurrence of association-deep and association-wide structures is of significance. This coincides with our findings.

This page intentionally left blank

9 Aggregate Functions & Rule Discovery

The experiments in the previous chapter demonstrate the power of aggregate functions. Particularly on databases that combine a high level of non-determinacy with large numbers of (numeric) attributes, these aggregate functions are a promising means of capturing local structure. Unfortunately, the propositionalisation method in which we embedded these functions has a disadvantage that reduces their potential. This has to do with the static nature of the features produced in the propositionalisation step. The small set of aggregate functions employed produces a moderate set of fixed features which cannot be modified when the actual mining takes place.

In contrast, the top-down algorithms presented in Chapters 5 and 6 select a new set of relevant features dynamically, based on the subgroup under investigation, say at a branch in a decision tree. As such, the range of substructures that can be recognised is not fixed from the outset. In this chapter we propose a method that combines the expressive power of aggregate functions with the desired dynamic behaviour of the top-down algorithms.

The aim is to include aggregate functions in the Rule Discovery framework introduced in Chapter 5. The selection graphs pattern language is extended to deal with our need for aggregation. Edges in the graph, which originally represented existential constraints, are annotated with aggregate functions that summarise the substructure selected by the connected subgraph. Although we continue to use the restricted set of aggregate functions, a large range of features can be considered by refining the subgraph with the usual conditions, and even with further aggregation-based conditions on the subgraph.

The overall structure of the Rule Discovery method remains unchanged. The primary changes deal with the richer pattern language and the refinement operator that introduces aggregate functions into a given pattern. We will show that extra care needs to be taken in order to guarantee that the refinement operator is actually a specialisation operator. Catering to aggregation also requires more complex data mining primitives. This subject is covered in Section 10.5.

Related work on aggregate functions primarily focuses on propositionalisation [59, 68, 69, 85]. To our knowledge, only a single method (relational probability trees [84]) introduces aggregate

functions dynamically. This method is currently limited to data models with an association-depth of 1, although the authors mention the possibility of extending the method. However, relational probability trees do not cater to aggregation in arbitrary locations in the data model, in the general way of our approach, nor do they allow nesting of aggregate functions.

9.1 Generalised Selection Graphs

In this section, we will show how aggregate functions are a natural generalisation of the existential conditions represented by edges in a selection graph. To support aggregate functions with selection graphs, we have to extend the language with a selection mechanism based on local structure. In particular, we add the possibility of *aggregate conditions,* resulting in *generalised selection graphs* (GSG).

Definition 9.1 An *aggregate condition* is a triple (f, θ, v) where f is an aggregate function, θ a comparison operator, and v a value of the codomain of f.

Definition 9.2 A *generalised selection graph* is a directed graph (N, E), where N is a set of triples (t, C, s), t is a table in the data model and C is a, possibly empty, set of conditions on attributes in t of type ($t.a$ θ c); θ one of the following operators, $=, \leq, \geq$. The flag s has the possible values *open* and *closed*. E is a set of tuples (p, q, a, F) where $p, q \in N$, a is an association between $p.t$ and $q.t$ in the data model, and F is an aggregate condition. The generalised selection graph contains at least one node n_0 (the *root-node*) that corresponds to the target table t_0.

Generalised selection graphs are simply selection graphs with two important extensions. First, the use of edges to express existential constraints is generalised by adding an aggregate condition. Secondly, nodes may be either *open* or *closed*, which determines the possible refinements (see next section). Note that the pattern language SG is contained in GSG. The concept of a selection edge is replaced by the equivalent aggregate condition (*count*, >, 0). In fact, we will use the directed edge without any aggregate condition as a purely syntactic shorthand for this aggregate condition.
A given generalised selection graph should be interpreted as follows. Every node n represents a set of records in a single table in the database. This set is governed by the combined restriction of the set of attribute conditions C and the aggregate conditions produced by each of the subgraphs connected to n. Each subgraph represents a similar set of records, which can be turned into a virtual attribute of n by the aggregate function. Consequently, we have a recursive definition of the set of records represented by the root-node. Each record corresponds to one individual that is covered by the graph.
We will use the same recursive construct in a translation procedure for SQL, as follows. If we start at the leaves of the graph and work back to the root, respecting all the selection conditions, we can compute the selection of records in the target table. This is achieved as follows. First we produce a list of groups of records in a table Q at a leaf node by testing on the aggregate condition. Each group is identified by the value of the foreign key:

```
SELECT foreign-key
FROM Q
WHERE  attribute-conditions
GROUP BY foreign-key
HAVING aggregate-condition
```

We then join the result *Q'* with the parent table *P* to obtain a list of records in *P* that satisfies the combined conditions in the edge and leaf node:

```
SELECT P. primary-key
FROM P, Q'
WHERE P. primary-key = Q'. foreign-key
```

This process continues recursively up to the root-node, resulting in a large query of nested `SELECT` statements. The second query can be extended with the grouping construct of the first query for the next edge in the sequence. This results in exactly one `SELECT` statement per node in the graph. This process is formalised in Figure 9.1.

```
SelectSubGraph (node n)

    S = 'SELECT ' + n.Name( ) + '.'
    if (n.IsRootNode( ))
        S.add (n.PrimaryKey( ))
    else
        S.add (n.ForeignKey( ))
    S.add (' FROM ' + n.Name( ))
    for each child i of n do
        S_i = SelectSubGraph(i)
        S.add(', ' + S_i + ' S' + i )
    S.add(' WHERE ' + n.AttributeConditions( ))
    if (!n.IsLeaf( ))
        for each child i of n do
            S.add (' AND ' + n.Name( ) + '.' + n.PrimaryKey( ) +
                ' = S' + i + '.' + i.ForeignKey( ))
    if (! n.IsRootNode( ))
        S.add(' GROUP BY ' + n.Name( ) + '.' + n.ForeignKey( ))
        S.add(' HAVING ' + n.ParentEdge( ).AggregateCondition( ))
    return S
```

Figure 9.1 The *SelectSubGraph* algorithm.

9.2 Refinement Operator

As Definition 2.13 states, a refinement operator performs purely syntactic operations. Most SDM algorithms employ a syntactic notion of generality: by simple syntactic operations on hypotheses, more specific clauses can be derived. For example, ILP often uses θ-subsumption [30, 103] for refinements, and our multi-relational rule discovery algorithm based on selection graphs employs ρ_{SG}. The two basic forms of operation are:

- add a condition to an existing node in the selection graph (roughly corresponds to a substitution in FOL).
- add a new node to the selection graph (add a literal to the clause).

It can be proved that such transformations constitute specialisations for SDM languages such as SG and FOL. For example, ρ_{SG} and refinement operators based on θ-subsumption are top-down refinement operators. However, this is not the case for the generalised selection graphs that we are

considering. Simple substitutions of the above-mentioned style do not necessarily produce more specific patterns.

Example 9.1 Consider the Mutagenesis database. Assume we are refining a promising hypothesis: the set of molecules with up to 5 atoms.

A simple substitution would produce the set of molecules with up to 5 C-atoms. Unfortunately, this is a superset of the original set. The substitution has not produced a specialisation.

The problem illustrated by Example 9.1 lies in the response of the aggregate condition to changes of the aggregated set resulting from added attribute conditions. Refinements to a subgraph of the generalised selection graph only work well if the related aggregate condition is *monotonic*.

Definition 9.3 An aggregate condition (f, θ, v) is *monotonic* if, given sets of records S and S', $S' \subseteq S$, $f(S')\ \theta\ v \Rightarrow f(S)\ \theta\ v$.

Lemma 9.1:

1. Let A = $(f, \geq, v)$ be an aggregate condition. If f is ascending, then A is monotonic.
2. Let A = $(f, \leq, v)$ be an aggregate condition. If f is descending, then A is monotonic.
3. The following aggregate conditions are monotonic: (*count*, $\geq$, v), (*count distinct*, $\geq$, v), (*max*, $\geq$, v), (*min*, $\leq$, v).

Lemma 9.1 shows that only a few aggregate conditions are safe for refinement. A subgraph of the generalised selection graph that is joined by a non-monotonic aggregate condition should be left untouched in order to only have refinements that reduce the coverage. The flag *s* at each node indicates whether nodes are safe starting points for refinements. *Closed* nodes will be left untouched.
This leads us to the following refinement operator ρ_{GSG} for GSG:

- **condition refinement.** An attribute condition is added to C of an *open* node.
- **association refinement.** This refinement adds a new edge and node to an *open* node according to an association and related table in the data model. An aggregate condition for the edge is specified. If the aggregate condition is monotonic, the new node is *open*, but *closed* otherwise.

The classes of refinements provided by ρ_{GSG} imply a manageable set of actual refinements. Each *open* node offers opportunity for refinements by each class. Each attribute in the associated table gives rise to a set of condition refinements. Note that we will be using primitives (Chapter 10) to scan the domain of the attribute, and thus naturally treat both nominal and numeric values. Each association in the data model connected to the current table gives rise to association refinements. The small list of SQL aggregate functions forms the basis for the corresponding aggregate conditions.

Proposition 9.1 The refinement operator ρ_{GSG} is a top-down refinement operator.

The proof works along the lines of Proposition 6.1 through 6.3. The monotonic behaviour of *present* edges in ESG is replaced by the monotonic aggregate conditions listed in Lemma 9.1. Additions to subgraphs only involve monotonic paths up to the root-node.

9.3 Experiments

The aim of our experiments is to show that we can benefit from using an MRDM framework based on GSG rather than SG. If we can show empirically that substantial results can be obtained by using a pattern-language with aggregate functions, then the extra computational cost of GSG is justified. In order to perform these experiments, we have upgraded the SGRules algorithm presented in Figure 5.1 to work with generalised selection graphs, and the extended refinement operator ρ_{GSG}. We will refer to this algorithm as GSGRules. The Rule Discovery algorithms offers a choice of rule evaluation measures of which two (novelty and accuracy) have been selected for our experiments. Rules are ranked according to the measure of choice. As search strategy, we have selected best-first search, because this has shown to produce dependable results during prior experiments.

9.3.1 Mutagenesis

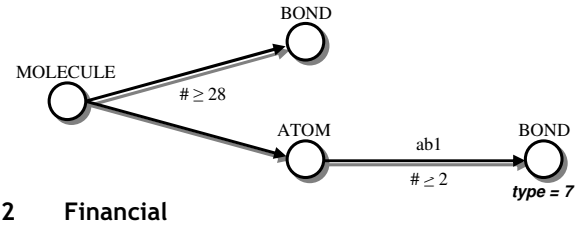
For the first experiment, we have used the Mutagenesis database with background B3. This database describes 188 molecules falling into two classes, *mutagenic,* and *non-mutagenic*, of which 125 (66.5%) are *mutagenic.* The description consists of the atoms and the bonds that make up the compound. In particular, the database consists of three tables that describe directly the graphical structure of the molecule (**molecule**, **atom**, and **bond**).

		novelty		accuracy	
		GSG	SG	GSG	SG
Best	novelty	0.173	0.146	0.121	0.094
	accuracy	0.948	0.948	1.0	1.0
	coverage	115	97	68	53
Top 10	novelty	0.170	0.137	0.115	0.088
	accuracy	0.947	0.914	1.0	1.0
	coverage	113.2	103.4	64.6	49.6

Table 9.1 Summary of results for Mutagenesis.

Table 9.1 presents a summary of the results. The left and right half give results when optimising for novelty and accuracy, respectively. The top half gives the measures for the best rule, whereas the bottom half gives the averages over the best 10 rules. We see that all results for GSG are at least as good as those of SG. In those cases were both SG and GSG have the maximum accuracy, GSG has a far larger coverage.

A typical example of a result is given below. This generalised selection graph describes the set of all molecules that have at least 28 bonds, as well as an atom that has at least two bonds of type 7. Clearly, this result cannot be found when using existential features solely, because of the condition on the count of bonds. It would also be difficult for propositionalisation approaches because there is a condition on the existence of an atom, followed by an aggregate condition on bonds of type 7 of that atom.



9.3.2 Financial

The second experiment concerns the Financial database. As in Section 5.3, we are looking for rules covering a minimum of 170 individuals (25%). Table 9.2 summarizes the results obtained. Clearly, all results for GSG are significantly better than those for SG, in terms of both novelty and accuracy. Admittedly, the results were obtained with rules of lower coverage, but this was not an optimisation criterion.

		novelty		accuracy	
		GSG	SG	GSG	SG
Best	novelty	0.052	0.031	0.051	0.025
	accuracy	0.309	0.190	0.314	0.196
	coverage	178	268	172	199
Top 10	novelty	0.050	0.027	0.047	0.023
	accuracy	0.287	0.178	0.297	0.185
	coverage	194.5	281.8	171.4	208.3

Table 9.2 Summary of results for Financial.

10 MRDM Primitives

10.1 An MRDM Architecture

Data Mining, and specifically the multi-relational variety, is a computation-intensive activity. Especially with serious applications of industrial size, a lot of data needs to be processed and scanned multiple times, in order to validate the large number of hypotheses generated by the algorithms such as covered in this text. We demand implementations of these algorithms to be at least moderately efficient and more importantly scalable, that is to perform predictively with increasing data volumes. These demands force us to pay attention to the architecture of such implementations. In this section we propose a three-tiered Client/Server architecture that satisfies these demands. It has been applied with success in Propositional Data Mining software [42, 46, 55].
An important subject in Data Mining architecture is how data access is organised. The remaining sections of this chapter are therefore devoted to multi-relational data mining primitives (see also Section 5.2) that address this subject.
The architecture consists of three tiers (see Figure 10.1): the *presentation tier*, the *Data Mining tier* and the *database tier*. Each tier will typically run on a separate workstation that is tuned towards the specific needs of the tier in question. For example the database tier will ideally run on a large server with ample main memory and fast access to secondary storage, whereas for the presentation tier, a regular desktop machine will suffice. Although separate workstations for each tier are optimal, several tiers can be run on a single machine. The presentation and Data Mining tier are an obvious choice, as they both have moderate computation requirements. The MRDM package Safarii follows this scheme.
The database tier has exclusive access to the data. It is responsible for the actual scanning of this data, and answering queries about the frequency of certain patterns. Furthermore its task is to find an efficient way of processing the many (similar) queries. The Data Mining tier is responsible for guiding the search, and producing interesting hypotheses for the database tier to test. It will often consist of an MRDM kernel that deals with the basics of MRDM, such as the representation of data models, selection graphs and refinements etc., as well as a number of search algorithms that

implement the different mining strategies. Finally, the presentation tier is responsible for interfacing with the user.

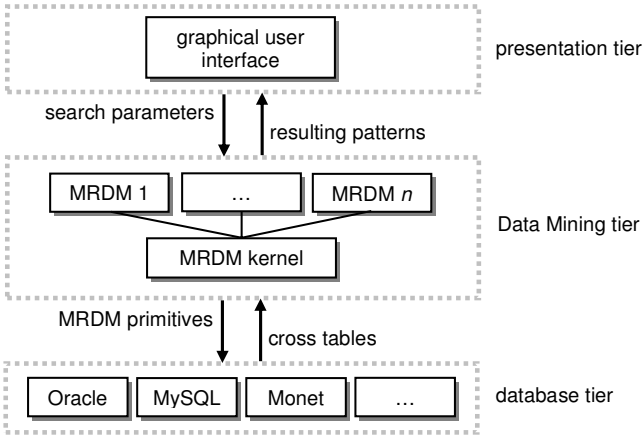


Figure 10.1 A three-tiered architecture for (Multi-Relational) Data Mining software.

Each tier has a limited responsibility and can be optimised for this specifically. Communication between tiers is relatively limited. The graphical user interface communicates with the data Mining algorithm only at the beginning and end of a run. The Data Mining tier communicates with the database tier through predefined data mining primitives. This amounts to sending a small query and receiving back a compact list of statistics that describes the coverage of a list of patterns, as implied by the query.

Because of the relative independence of tiers, parts of the overall system can be replaced or optimised without interference with the remaining components. Specifically the database tier allows a range of RDBMSs to be employed. By using a vendor-independent database connectivity layer, such as ODBC or JDBC, to communicate between the database and Data Mining tier, alternative databases may be tried without affecting the other components.

Because the hypothesis testing is separated from the mining process, and organised into primitives, the RDBMS is allowed to optimise the data access, for example by parallelising the query processing. Also, queries stemming from different Data Mining tools working on the same data could be integrated. An interesting development is this respect is the work by Manegold et al. [76] on the database system Monet [15]. They observe that due to the nature of top-down (Propositional) Data Mining algorithms, many of the data mining primitives are concerned with very similar collections of data. By eliminating common sub-expressions in the query-graphs, and buffering partial results of previous queries, the query processing can be optimised considerably. This method works in a manner that is transparent to the Data Mining tier, and can thus easily be inserted in place of more mainstream RDBMSs.

10.2 Data Mining Primitives

The use of data mining primitives in Data Mining is quite widespread [37, 42, 50]. They are a means of organising the access to the data that is necessary for obtaining statistical information about the pattern under investigation and possible refinements thereof. Rather than incorporating

random access to individual pieces of data in the Data Mining algorithm, a small set of queries is defined, typically in SQL, that produce the necessary summaries to guide the search algorithm. The following sections provide an overview of the different multi-relational data mining primitives required to support the variety of MRDM approaches introduced in the previous chapters.
Data mining primitives have a number of important benefits:

- they represent a concise summary of both the necessary and sufficient information required to judge a pattern. They provide the efficient communication that is typical of a good Client/Server architecture.
- a range of possible refinements for a given pattern is investigated by a single primitive. The RDBMS is thus given an opportunity to optimise multiple very similar scans of the data.
- deciding which actual refinements will be considered by the search process is left to the RDBMS to some extent. When considering, for example, a condition refinement on a given selection graph for a given node and attribute, the primitive will list all the values that are available for this attribute under these circumstances. This allows algorithms to consider the correct set of condition refinements, without *a priori* knowledge of the domain of the attribute. Particularly with aggregate functions, where the domain of the function depends entirely on the aggregated set, a data-driven generation of possible aggregate conditions is essential.

Although a large variety of primitives can be defined, we will be using primitives that produce counts. These counts will be in terms of individuals, i.e. in terms of records in the target table. Each count thus represents the coverage of a specific subgroup.
We will be considering primitives in two styles. In the first style, we will be treating selection graphs as specifications of joins over the tables involved. Because we can think of both the existential constraints as well as the attribute conditions as selections on the Cartesian product of the tables, the whole selection can be expressed in a single `SELECT` statement. This will produce SQL statements similar to those introduced in Chapter 4. The second style of primitives is more applicable if patterns involve aggregate functions. Here we treat (generalised) selection graphs as recursive definitions of groups of records associated with each node. We will be using nested queries rather than joins to produce the collective counts over individuals.
In the next sections we will consider primitives for each refinement, for selection graphs, extended selection graphs, and generalised selection graphs in turn.

10.3 Selection Graphs

The following four data mining primitives can be used to support the discovery of selection graphs as defined in Chapter 4. The first two primitives provide the necessary counts to test association refinements. The next pair supports condition refinements.

10.3.1 Association Refinement

The following primitive counts the number of individuals covered by a given selection graph. It basically counts over a selection of the Cartesian product of tables involved (see the *TranslateSelectionGraph* algorithm in Section 4.2). As individuals can occur multiple times in this selection, we have to apply the `DISTINCT` operator. Although the CountSelection primitive can be used to compute the coverage of arbitrary selection graphs, we will mainly use it to assess the interestingness of an association refinement by running the primitive on the selection graph produced by the association refinement. This count is typically compared to the coverage of the selection graph prior to refinement, or to that of the empty selection graph.

CountSelection:

SELECT COUNT(DISTINCT t_0.*primary_key*)
FROM *table_list*
WHERE *join_list* AND *condition_list*

Instead of working with the coverage of subgroups, we will often be interested in the distribution of values for the target attribute within the subgroup. From these counts, and the overall distribution, we can compute the value of a range of evaluation measures, as was briefly demonstrated in Section 5.2. The Histogram primitive produces this list of counts for association refinements. Note that the sum of the counts in the resulting histogram is equal to the result of the CountSelection primitive.

Histogram:

SELECT t_0.*target*, COUNT(DISTINCT t_0.*primary_key*)
FROM *table_list*
WHERE *join_list* AND *condition_list*
GROUP BY t_0.*target*

10.3.2 Nominal Condition Refinement

Similar to the Histogram primitive, the NominalCrossTable primitive produces counts pertaining to the distribution of values for the target attribute. In this case however, we are interested in how this distribution depends on the value of a nominal attribute $t_i.c_j$ in one of the tables appearing in the selection graph at hand. The NominalCrossTable primitive will list all pairs of values that occur in the database, along with the number of individuals this combination occurs in. The resulting cross table can be used to compute a range of statistical measures describing the dependency between these two attributes. Note that the sum of these counts can exceed the support of the given multi-relational pattern, if the attribute $t_i.c_j$ is in a table that is not the target table. This is because multiple records, with different values for the selected attribute, may correspond to a single record in the target table, causing this record to contribute to multiple counts in the cross table.
The NominalCrossTable primitive can be used to evaluate a set of condition refinements involving the condition $t_i.c_j = x$. Each count in the cross table describes the number of individuals for a particular target value, as if the condition refinement were already executed for a particular value of $t_i.c_j$. The primitive will only return combinations of values that actually appear in the database.

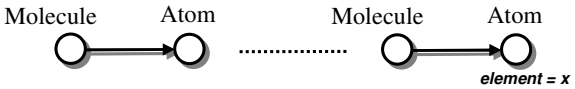
NominalCrossTable:

SELECT t_0.*target*, $t_i.c_j$, COUNT(DISTINCT t_0.*primary_key*)
FROM *table_list*
WHERE *join_list* AND *condition_list*
GROUP BY t_0.*target*, $t_i.c_j$

Example 10.1 Consider the following NominalCrossTable primitive introduced in Section 5.2:

```
SELECT molecule.mutagenic, atom.element,
     COUNT(DISTINCT molecule.id)
FROM molecule, atom
WHERE molecule.id = atom.molecule_id
GROUP BY molecule.mutagenic, atom.element
```

This query supports a set of tentative condition refinements on the *element* attribute in **atom**, as follows. Based on the counts for different values of x and the associated evaluation measure, the Data Mining algorithm will select a number of values for refinement:



10.3.3 Numeric Condition Refinement

The NumericCrossTable primitive can be used to compute the dependency between a numeric attribute and the target attribute. We assume that each record in the target table has multiple associated records in table t_i. For each of these sets of records in table t_i we can compute the minimum for the attribute of interest $t_i.c_j$. The primitive produces a list of counts for pairs of values for the target attribute and each occurring minimum. The occurring minimums will be used to produce a list of candidate tests on the attribute $t_i.c_j$, as outlined in Section 4.4. An analogous call for the maximum exists. Note that the NumericCrossTable primitive will typically return far fewer values than occur in $t_i.c_j$, as only the minimum or maximum per individual are considered.
The NumericCrossTable primitive can be used to evaluate a set of condition refinements involving the condition $t_i.c_j \leq x$ (or $t_i.c_j \geq x$). Each count represents the number of individuals for a particular target value and minimum (maximum) value. By considering these counts in cumulative form – progressively adding counts – we obtain counts for different subgroups, as if the condition refinement involving $t_i.c_j \leq x$ ($t_i.c_j \geq x$) were already executed for a particular value of $t_i.c_j$

NumericCrossTable:

```
SELECT t0.target, m, COUNT(*)
FROM
     (SELECT t0.target, t0.primary_key, MIN(ti.cj) m
      FROM table_list
      WHERE join_list AND condition_list
      GROUP BY t0.target, t0.primary_key)
GROUP BY t0.target, m
```

Example 10.2 Consider the following NumericCrossTable primitive for the numeric attribute *type* in the **atom** table:

```
SELECT mutagenic, m, COUNT(*)
FROM
     (SELECT molecule.mutagenic, molecule.id,
             MIN(atom.type) m
      FROM molecule, atom
      WHERE molecule.id = atom.molecule_id
      GROUP BY molecule.mutagenic, molecule.id)
GROUP BY mutagenic, m
```

This primitive supports refinements of the following kind:

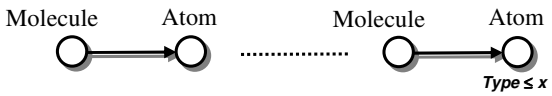


Figure 10.2 shows the outcome of this primitive (top table). Table headers are values for *type* that turn out to be thresholds. Empty cells represent combinations that do not appear in the database. In cumulative form (bottom table), we obtain counts for subgroups with *type* ≤ *x*. For example, 54 + 41 = 95 molecules contain an atom of *type* ≤ 16, and a rule involving *mutagenic* = T has a novelty of – 0.0487.

	1	10	14	16	21	22	194	195
T	17	34	3		7	54	1	9
F	20	16	2	3		22		

	1	10	14	16	21	22	194	195
T	17	51	54	54	61	115	116	125
F	20	36	38	41	41	63	63	63

Figure 10.2 Results of NumericCrossTable in normal and cumulative form.

10.4 Extended Selection Graphs

Obtaining the necessary counts for extended selection graphs is very similar to the process described in the previous section. The same set of four primitives can be used to produce counts for the 'positive' refinements. For the complementary refinements we can simply subtract the positive counts from the basic counts for the pattern under consideration. These basic counts can be obtained by running a CountSelection or Histogram primitive on the pattern at hand, without refinements, or by storing the counts when they were first computed when the candidate pattern was produced. In this way, counts for complementary refinements can be obtained without having to access the database.

Example 10.3 Consider the NominalCrossTable primitive presented in Example 10.1 (see also Section 5.2). This primitive contains counts for a collection of condition refinements for the *element* attribute on a (extended) selection graph involving **molecule** and **atom**. By subtracting the NominalCrossTable primitive from the counts for the empty graph ([125, 63]), we obtain the following counts for the collection of complementary condition refinements. The value 0 represents a combination that only appears in the 'positive' subgroup.

	Br	C	Cl	F	H	I	N	O
T	124	0	122	121	0	124	0	0
F	62	0	55	58	0	63	0	0

When producing the SQL statement pertaining to the primitive at hand, we will have to incorporate the *absent* edges that may occur in extended selection graphs. The translation procedure can simply use the *TranslateSubgraph* function to obtain a *join_list* which expresses any occurring *absent* subgraphs, analogous to the SQL translation described in Chapter 6.
The primitives for ESG will furthermore have to accommodate the look-ahead refinements (see Section 6.3), as follows. The 'positive' look-ahead condition refinement consists of an association refinement followed by a condition refinement. A NumericCrossTable or NominalCrossTable primitive that already incorporates the first association refinement in the *table_list* and *join_list* will produce the desired result. Complementary counts can be obtained as before. In a similar way, the

look-ahead association refinement can be validated by a Histogram primitive that includes the two new tables in the *table_list* and *join_list*.

10.5 Generalised Selection Graphs

We start by repeating the basic process of recursively selecting groups of records, as described in Chapter 9 and Figure 9.1. This process will be the basis for the primitives related to generalised selection graphs. Every node in a generalised selection graph represents a selection of records in the associated table. If we start at the leaves of the graph and work back to the root, respecting all the selection conditions, we can compute the selection of records in the target table. This is achieved as follows. First we produce a list of groups of records in a table Q at a leaf node by testing on the aggregate condition. Each group is identified by the value of the foreign key:

```
SELECT foreign_key
FROM Q
WHERE attribute_conditions
GROUP BY foreign_key
HAVING aggregate_condition
```

We then join the result Q' with the parent table P to obtain a list of records in P that adheres to the combined conditions in the edge and leaf node:

```
SELECT P. primary_key
FROM P, Q'
WHERE P.primary_key = Q'. foreign_key
```

This process continues recursively up to the root-node, resulting in a large query C of nested `SELECT` statements. The different primitives are all variations on this basic construct.

10.5.1 Association Refinement

The CountSelection primitive simply counts the number of individuals covered by the generalised selection graph, as in the case of selection graphs:

CountSelection:

```
SELECT COUNT(*)
FROM C
```

The Histogram primitive computes the distribution of values of the target attribute within the set of individuals covered by the generalised selection graph, as before.

Histogram:

```
SELECT target, COUNT(*)
FROM C
GROUP BY target
```

10.5.2 Nominal Condition Refinement

As before, the NominalCrossTable primitive can be used to determine the effect of a condition refinement involving a nominal attribute on the distribution of target values. All pairs of nominal

values and target values are listed with the associated count. As the nominal attribute typically does not appear in the target table, we cannot rely on the basic construct to produce the NominalCrossTable. Rather, we will have to augment the basic construct with book-keeping of the possible nominal values.
The idea is to keep a set of possible selections, one for each nominal value, and propagate these selections from the table that holds the nominal attribute, up to the target table. These selections all appear in one query, simply by grouping over the nominal attribute in the query that aggregates the related table. The basic construct is extended, such that an extra grouping attribute *X* is added to all the nested queries along the path from the root-node to the node that is being refined:

```
SELECT X, foreign_key
FROM Q
GROUP BY X, foreign_key
HAVING aggregate_condition
```

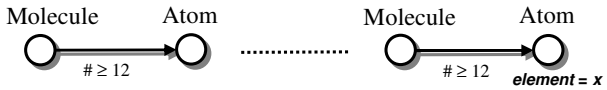
The extra attribute in this query will reappear in all (nested) queries along the path to the root-node. The actual counting is done in a similar way as for the Histogram:

```
SELECT X, target, COUNT(*)
FROM C
GROUP BY X, target
```

Example 10.4 Assume we are considering molecules that contain at least 12 atoms. The following Histogram query produces the desired counts, arranged by the values T and F for the target attribute. It turns out that all molecules contain at least the desired 12 atoms.

```
SELECT molecule.mutagenic, COUNT(*)
FROM molecule,
     (SELECT molecule_id
      FROM atom
      GROUP BY molecule_id
      HAVING COUNT(*) >= 12) c
WHERE molecule.id = c.molecule_id
GROUP BY molecule.mutagenic
```

In order to test candidate condition refinements on **atom**, we run a NominalCrossTable primitive for the nominal attribute *element*.



This primitive lists counts for subgroups of molecules containing at least 12 atoms of a particular element, for each value of the target attribute.

```
SELECT molecule.mutagenic, c.element, COUNT(*)
FROM molecule,
     (SELECT element, molecule_id
      FROM atom
      GROUP BY element, molecule_id
      HAVING COUNT(*) >= 12) c
WHERE molecule.id = c.molecule_id
GROUP BY c.element, molecule.mutagenic
```

This query produces the following results. For example, there is a single non-mutagenic molecule that has at least 12 hydrogen atoms. Note that only carbon and hydrogen appear frequent enough per molecule to appear in the result.

	C	H
T	112	12
F	18	1

10.5.3 Numeric Condition Refinement

As before, the NumericCrossTable primitive can be used to obtain a list of suitable numeric refinements together with the effect this has on the class-distribution. It is very similar to the NominalCrossTable, but requires yet a little more book-keeping. This is because a given record may satisfy a number of numeric conditions, whereas it will only satisfy a single nominal condition. The process starts by producing a list of combinations of records and thresholds, and then proceeds as before. Note that there is a version of the NummericCrossTable for each operator ≤ and ≥.
The NumericCrossTable primitive for GSG works differently from its SG counterpart. Because the GSG primitives do not work on selections of the Cartesian product, we can not rely on the minimum and maximum as before. Rather we will have to keep track of different tentative numeric thresholds and propagate the associated selections of records up to the target table.

```
SELECT b.X, a.foreign_key
FROM Q a, (SELECT DISTINCT X FROM Q) b
WHERE a.X <= b.X
GROUP BY b.X, a.foreign_key
HAVING aggregate_condition
```

Example 10.5 Assume we are looking for subgroups of molecules that contain at least 12 atoms again, and are interested in the effect of adding a numeric condition on the atoms. The following primitive produces counts for condition refinements involving **atom**.*type* ≤ *x*. The query lists suitable values for *x*, along with the number of individuals that satisfy these conditions.

```
SELECT molecule.mutagenic, c.type, COUNT(*)
FROM molecule,
     (SELECT b.type, a.molecule_id
      FROM atom a, (SELECT DISTINCT type FROM atom) b
      WHERE a.type <= b.type
      GROUP BY b.type, a.molecule_id
      HAVING COUNT(*) >= 12) c
WHERE molecule.id = c.molecule_id
GROUP BY c.type, molecule.mutagenic
```

This NumericCrossTable primitive produces the following results. Values above the columns indicate thresholds for *type*.

	3	8	10	14	16	19	21	25	26	27	…
T	12	12	20	20	23	23	28	119	119	121	…
F	1	1	4	4	6	6	6	50	52	52	…

10.5.4 AggregateCrossTable

The AggregateCrossTable is our most elaborate primitive. It can be used to obtain a list of suitable aggregate conditions for a given aggregate function. The AggregateCrossTable primitive demonstrates the power of data mining primitives because the list of suitable conditions can only be obtained through extensive analysis of the database. It cannot be obtained from the domain knowledge or a superficial scan when starting the mining process.
We will treat candidate aggregate conditions as virtual attributes of the parent table. A first query is applied to produce this virtual attribute. Then the NumericCrossTable primitive is applied with the virtual attribute as the candidate. The first step looks as follows:

```
SELECT foreign_key, aggregate_function
FROM Q
GROUP BY foreign_key
```

Example 10.6 Assume we are working with the subgroup of molecules that contain at least 12 atoms, as in Example 10.4 and 10.5. If we are considering an association refinement involving counting over one of the associations between **atom** and **bond**, we can apply the above query. This query produces a virtual numeric attribute in **atom**, representing the number of associated bonds:

```
SELECT atom1, COUNT(*) c
FROM bond
GROUP BY atom1
```

We can now run the NumericCrossTable primitive, as we did before on the *type* attribute.

11 MRDM in Action

For the bulk of this thesis, we have focussed on the essential elements and basic algorithms of MRDM. These tools are crucial for a data analyst, but not sufficient for the successful completion of an MRDM project. From the point of view of the analyst, the actual algorithmic details will often be of less importance than the preparatory activities preceding the actual analysis step. In this chapter, we consider MRDM from a higher, more practical level and provide a blueprint for MRDM projects, as well as technology to support such projects.

11.1 An MRDM Project Blueprint

A data analyst is concerned with producing optimal results from a given dataset. In the majority of cases, this is not simply a matter of applying the algorithm of choice to the database as is. There is a fair amount of effort involved in understanding the domain the data describes. More importantly, the data will typically be in a format optimised for transaction processing, and a substantial amount of massaging of the data will be necessary to make it fit for multi-relational analysis. Often, this exploration of the domain and preparing of the data will take up more time than the analysis itself. As this phase of a Data Mining project is non-trivial, especially where MRDM is concerned, a project blueprint detailing the different steps is justified. Figure 11.1 shows the outlines of this blueprint. On a global level, the blueprint follows the structure of the CRISP-DM Process Model [18], a standard for approaching Data Mining problems in general.

Apart from the Modelling task, two important tasks will be described in detail, because multi-relational issues play a major role here: Data Understanding and Data Preparation. Data Understanding deals with gathering meta-data from the database and alternative sources, and deciding how to best approach the analysis. The Data Preparation task is concerned with transformation of the database in order to obtain optimal results for MRDM in general, and selected algorithms in specific. The subtasks of these two tasks are listed in Figure 11.2. The remaining tasks are mainly representation-independent, and are therefore of less importance to the current discussion. They deal with the high level definition of the goals of the project (Business

Understanding task), whether these goals are sufficiently addressed by the models produced (Evaluation task), and how the knowledge gained can be integrated in the business process (Deployment task).

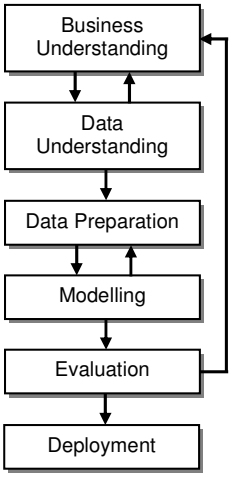


Figure 11.1 The different tasks of the MRDM project blueprint.

Data Understanding	**Data Preparation**
define individual	construct features
define target concept	unfold recursion
determine background	discretise numeric data
specify test set-up	denormalise
select tables and attributes	aggregate and propositionalise
add directions	

Figure 11.2 The data Understanding and Data Preparation task in detail.

The subtasks of the Data Understanding and Data Preparation task entail the following activities:

Define individual An essential step in any MRDM project. Obvious candidates for the individual are the entities already present in the data model. In some cases a new entity must be distilled from existing ones.

Define target concept Many Data Mining projects include a target concept. In our MRDM framework, this target (if present) must appear as an attribute of the individual, i.e. as an attribute in the target table.

Determine background In order to better understand the structure of the data model, and decide on further steps in the project, the foreground, background and individual-data is determined. Section 3.3.1 outlines the details of this.

Specify test set-up Any serious Data Mining project involves validating the quality of the models. Validation is typically achieved by splitting the database into multiple samples, then building and testing the model on separate samples, referred to as the *training* and *test set.* The most common procedure, *n*-fold cross-validation, involves *n* mutually exclusive samples. The Data Mining algorithm is then applied *n* times to the union of *n*-1 samples with the resulting model tested on the remaining sample.
Clearly, any sampling in MRDM should be done on the level of the individual. This can easily be done if the foreground and the individual-data coincide. Care will have to be taken if tables appear in the foreground that are not individual-specific. This dynamic data that is shared by samples can best be removed from the analysis.

Select tables and attributes Selecting a subset of the available information for analysis is a simple, yet important and frequent operation in Data Mining projects. It is mainly used as it is typically used in Propositional Data Mining: to restrict the scope of the analysis, to reduce the analysis time, to remove information that directly depends on the target concept, or to remove data that is shared by individuals.

Add directions Section 3.3.2 explains how directions can be an important form of declarative bias that can help make the search more efficient or relevant.

The Data Preparation task consists of the following subtasks:

Construct features A database designed for transaction processing will generally not contain any redundancy. For analysis however, having multiple views on the same information may be valuable because not all of these alternative representations may demonstrate the desired dependencies. Feature construction involves the definition of new attributes (or even tables) as a function of existing data, with the aim of providing the Data Mining tool with such alternative views. Feature construction is an important means of incorporating domain knowledge into the analysis. As an example, think of a database of objects in the physical world, where derived attributes such as surface area, volume or density can be just as informative as the primary dimensions of the objects. The validity of specific arithmetic functions of attributes must be made specific.

Unfold recursion A special form of feature construction deals with recursive structures. Recursion is an important challenge and research topic in ILP. Most methods aim at discovering opportunities for applying recursive patterns in databases of which little is know about the data model. In MRDM however, a detailed data model is generally available, and recursive structures in the data can be easily identified by locating possible cycles in the data model. Because our MRDM framework does not support recursive patterns, the unfold recursion subtask involves eliciting essential information from the recursive structure. The extracted features capture as much of the original information, while respecting the limitations of non-recursive algorithms.
Recursive fragments of a data model typically represent datastructures such as lists, trees or sets. Although the data representation is complete, it hides a number of properties of the datastructure (such as its size) to general purpose Data Mining algorithms. The aim of the unfold recursion task is to transform the data, such that important and promising features are made explicit, at the cost of losing some of the complete recursive description. This entails defining a number of new (ideally intensional) tables that replace the recursive fragment. Because these definitions need to consider the recursive structure, we will have to use a procedural language or, often vendor-specific, recursive constructs in SQL. Full support for recursion as included in the SQL3 definition is currently not provided by the major RDBMSs [101].

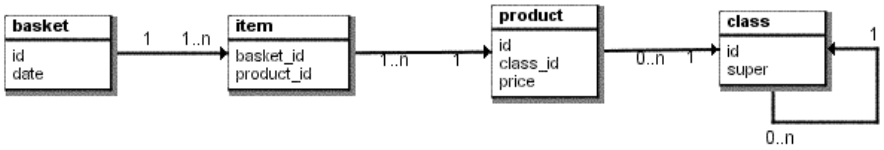
Discretise numeric data Section 4.4.1 describes how numeric attributes can be replaced by a set of binary attributes representing various numeric thresholds. We stress again, that discretisation introduces a substantial imprecision, and should therefore only be used to facilitate the use of algorithms with limited support for numeric data.

Denormalise Data models will often contain so-called *domain* or *look-up* tables: tables that are connected to the remaining tables via many-to-one associations exclusively. These tables provide background information to the referring tables, and are typically the result of normalisation during database design. For analysis purposes, joining in this data (denormalising) can be attractive as it removes tables from the data model.

Aggregate and Propositionalise If we intend to use a propositional Data Mining tool, we need to propositionalise the database at hand. A number of procedures is available for this task, roughly falling into two categories: procedures that produce a table of binary features [6, 24, 67, 65], each representing some interesting multi-relational pattern, and procedures that employ aggregate functions as described in Chapter 8.
Even when using an MRDM algorithm, one can still benefit from partial propositionalisation of the data. By applying aggregate functions on the level of pairs of individual tables, the structural information between these tables is enriched. MRDM algorithms can thus benefit both from traditional existential features, as well as aggregated features.

11.2 An MRDM Project

The purpose of this section is to demonstrate the proposed MRDM methodology on a possible database. We will consider a hypothetical retail-database and show the result of the different steps. The database consists of descriptions of shopping-baskets along with product information as well as a product-hierarchy. The data model below shows the four tables involved. Table **basket** identifies individual baskets, and is associated with the contents of the baskets stored in **item**. Every item is an instance of a **product**. In turn, every product belongs to a **class**, and classes are organised in a hierarchy.



11.2.1 Data Understanding

Because we are interested in the buying behaviour of customers, relating products bought together, we assign **basket** as our target table. If we are merely interested in association rules involving product and product-classes, we need not define a target concept. If, on the other hand, we have a particular product in mind, and want to explain or predict buying behaviour for it, we can create a feature in **basket** based on occurrence of this product in the basket.
There is a clear distinction between the foreground and background of the database. **Basket** and **item** belong to the foreground, because they contain dynamic data that changes constantly (at least in the operational system). The background is formed by **product** and **class**. The individual-data consists of **basket** and **item** (no item belongs to two baskets). **Product** and **class** are clearly not

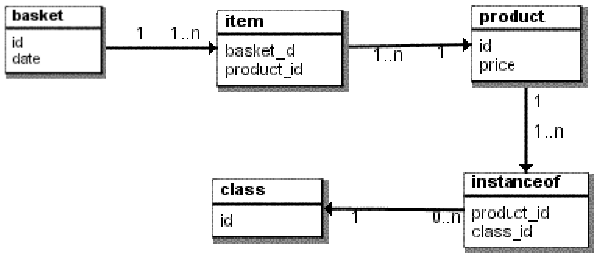
individual-data: the records in these tables are shared extensively. The individual-data coincides exactly with the foreground, indicating that we can safely sample on **basket**. We can also note that although the data model contains four tables and four associations, the actual structural information that distinguishes baskets is limited to the two tables **basket** and **item**.
For all associations, we define direction-restrictions:

- from **basket** to **item**. Although not strictly necessary, we exclude repetitive references to the same basket once items are introduced in patterns.
- from **item** to **product**. This restriction guarantees that item-information is not shared between baskets.
- from **product** to **class**. We choose to exclude information about other products in a class to which a product under consideration belongs.
- from **class** (child) to **class** (parent). This determines that we can introduce parents of a class that already occurs in a particular hypothesis, but not vice-versa.

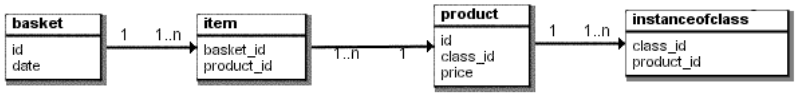
These directions were already indicated in the figure above.

11.2.2 Data Preparation

As we intend to use an MRDM tool that does not support recursion, we have to decide how to deal with the recursive association between classes. We decide that the actual parent-child relation can be ignored, but information about all classes a product belongs to (recursively) could be relevant. We introduce a new table **instanceof** that represents the many-to-many relationship between products and classes as defined by the class-hierarchy:

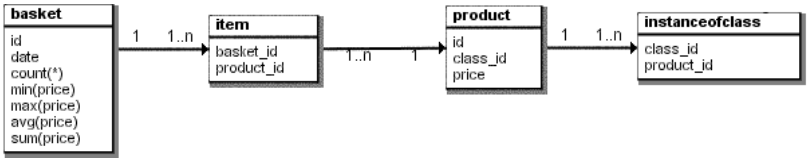


During the unfold recursion step, we have produced a many-to-one association with a direction-restriction, the result of earlier choices. We can simply join **instanceof** and **class**. In fact, this just removes the **class** table, because it contains no non-key attributes. The association between **item** and **product** is of a similar nature. After considering the number of items compared to the number of products however, we decide not to use an excessive amount of memory (or disk space) by denormalising, and thus replicating product-information.



If we intend to apply an MRDM algorithm that does not fully support numeric data, we will have to discretise *price* in **product**. We start by denormalising *price* on the **item** table. Then we compute the minimum and maximum price for each basket, producing two sets of numeric values. A regular

discretisation procedure is applied to each set, to produce two collections of, say, four thresholds, one for ≤ and one for ≥, respectively. The item table is now enriched with eight binary attributes corresponding to the eight thresholds.
If we choose to apply an algorithm that does support full treatment of numeric data, we can still consider some useful transformations based on the *price* attribute in **product**. Propositionalisation by means of aggregate functions (and denormalisation) will produce a number of valuable features on the basket level, such as the number of items in the basket, or the total price. Such a transformation produces the following data model:



11.3 An MRDM Pre-processing Consultant

Practical experience with MRDM projects shows that a lot of time, specifically man-hours rather than CPU-cycles, is spent on the tasks preceding the actual mining phase. In this and the following sections, we present a system called ProSafarii that automates a number of common operations in preparing multi-relational databases. The system focuses specifically on the tedious and laborious tasks, thus making the job of the data analyst more interesting and effective. By acting as an MRDM pre-processing consultant, ProSafarii even allows professionals with a less technical background to complete a full MRDM project.
ProSafarii aims at automating, at least to a certain extent, the following types of activities:

- recognising opportunities for transformation of the data model or database.
- performing the exploratory data analysis necessary to asses the applicability of certain tentative transformations.
- judging the desirability of transformations on the basis of data-driven heuristics and listing the advantages and disadvantages. This functionality is currently only present in rudimentary form.
- performing the actual database queries that implement the transformation, as well as the modifications to the data model that feeds into Safarii.

ProSafarii primarily focuses on Data Preparation tasks rather than Data Understanding tasks. It supports a collection of transformation rules that it will test against the initial data model, in order to discover opportunities for improvement. Transformed databases and data models then give rise to potential further transformations. As such, ProSafarii is able to perform a limited form of reasoning about multi-relational data models. As input for this process, it will take the graphical structure of the data model along with the actual content of the database. ProSafarii is semi-automated in that it will list instances of transformation rules and require the initiative of the user to actually execute a rule. Subsequent transformation rules will therefore only become available after completing the first transformation. When the user selects a rule instance for execution, they will, depending on the type of rule, have to specify a small number of options, such as the name of a new table, etc., before proceeding.
Figure 11.3 shows a screenshot of ProSafarii with a tree of tentative transformations on the left, organised by transformation rule class. On the right the current data model, depending on the selection made on the left-hand side, is visualised. ProSafarii integrates with Safarii through MRML files and shared connections to the database. Any database and data model produced by ProSafarii

can thus be mined directly using Safarii. The system will, in most cases, allow the execution of rules in one of three modes:

- new tables are extensional: actual database tables are created.
- new tables are intensional: only views are created. This only applies if the database in question supports views, and the intensional data can be expressed in a limited number of SQL statements.
- new tables are virtual: the new data model only exists as an MRML file. Only a limited set of subsequent transformation rules can be considered, as there is no actual data to assess their suitability.

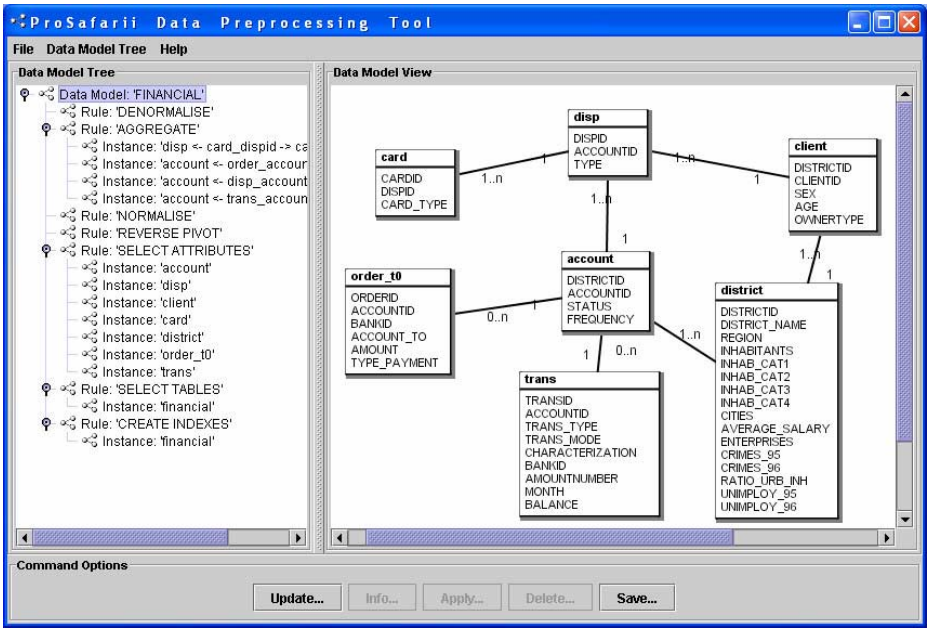


Figure 11.3 The ProSafarii pre-processing consultant.

11.4 Transformation Rules

In this section we list the collection of transformation rules currently supported by ProSafarii. This collection is biased towards Safarii, but can easily be extended to accommodate other Data Mining or data analysis tools.

11.4.1 Denormalise

The *denormalise* transformation joins two tables that are related through a many-to-one association, effectively attaching the information on the *one* side to the table on the *many* side.
The rule applies to two tables *A* and *B* under the following conditions:

- there is an association between *A* and *B* with multiplicity 1 on the *B* side.
- there is a single direction-restriction, from *B* to *A*.

Denormalisation is attractive because it removes one table from the data model. In some cases it will no longer be necessary to apply a multi-relational algorithm. In other cases, it will remove an

association, which reduces the necessary refinement-depth. A drawback is that the resulting table typically takes more space, as information from B is duplicated. For tables A and B of m and n records ($m > n$), and p and q attributes, the join will consist of m records and $p+q-1$ attributes.

Example 11.1 The tables **account** and **district** are good candidates for denormalisation. The detailed demographic data on 77 districts will be duplicated to extend the information for each of the 682 accounts. The new table will contain 682·19 = 12958 fields compared to the original 77·16 + 682·4 = 3960 fields.

11.4.2 Normalise

The *normalise* transformation aims to reduce redundancy in a table by splitting it into two. As such, it is the reverse of the denormalise transformation. A table exhibits redundancy if there is a specific form of dependency between attributes, known as a functional dependency. Tables with redundancy are frowned upon in relational database design, as they waste space and introduce update anomalies when modifying the database. Such redundancy is therefore often removed in a process called normalisation. Although well-designed databases will not allow further normalisation, denormalised tables are not uncommon in data analysis settings for reasons explained in the previous section.
A functional dependency is a dependency between two sets of attributes. A functional dependency $X \rightarrow Y$ holds in a table A, if for all records r and s such that $r[X] = s[X]$, it is also the case that $r[Y] = s[Y]$. If $X \rightarrow Y$ holds, then we can remove attributes Y, and create a new table containing attributes XY. In this case, we will only consider single-attribute keys, so X will have one element x. In order to test a specific functional dependency $x \rightarrow y$, the following SQL query is used:

```
SELECT MAX(c)
FROM
     SELECT COUNT(DISTINCT y) c
     FROM A
     GROUP BY x
```

If the result equals 1, $x \rightarrow y$ holds. All functional dependencies with the same x can now be grouped together to form $x \rightarrow Y$. A rule involving x and Y applies under the following conditions:

- $x \rightarrow Y$ holds
- x is not numeric, unique, or a primary key.

The respective advantages of both normalised and denormalised data models have been discussed in the previous sections. An important use of normalise is not to transform the data, but to better understand its structure. A denormalised data model may be beneficial for Data Mining, but more importantly, knowledge about the functional dependencies will also aid the correct interpretation of results.

Example 11.2 The Mutagenesis database contains a functional dependency *type*→ *element* in the **atom** table. Figure 11.4 shows the result of applying normalise.

11.4.3 Reverse Pivot

The *reverse pivot* rule applies to very long and narrow tables that represent attribute-value pairs. The result of the transformation is a shorter, but wider table that contains a collection of new attributes that were previously stored as data in the original table. In other words, it produces a propositional dataset by collecting several short records into a single long record. The attributes are effectively 'pivoted' from records to columns.

The reverse pivot transformation involves three attributes in different roles. A first attribute x groups records together and serves as the primary key of the new table. A second attribute y lists the attributes of the new table. A third attribute z specifies the value of the attribute y of record x in the new table.

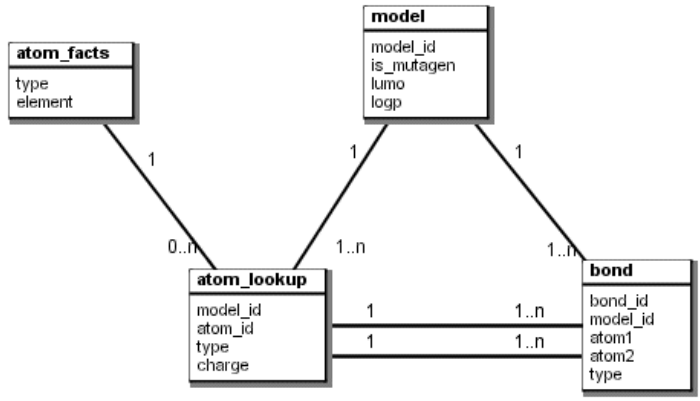


Figure 11.4 The result of normalise.

The reverse pivot rule applies under the following conditions. Let N be the number of records, and $|x|$, $|y|$, and $|xy|$ be the number of distinct values of x, y and the concatenation of x and y respectively.

- x and y are nominal.
- $|x| < N$, $|y| < N$. If x or y is unique, the resulting table will be impractically long or wide and the transformation will be useless.
- $|xy| = N$. For any given x, an attribute y may have only one value.
- $N / (|x| \cdot |y|) > \alpha$, $0 < \alpha < 1$. The fraction of fields in the new table that have a value should exceed a user-specified threshold α. α will typically be large, e.g. 0.9 or even 1.

Note how x and y can be interchanged in the conditions above. ProSafarii will always report two transformation rules for each suitable pair. The user will have to select the correct one. A rule of thumb in this selection is $|x| > |y|$. The user will furthermore have to specify which attribute takes the role of z.

It is clear that the reverse pivot operation is very useful for propositionalising a database, and thus facilitating the application of a wider range of Data Mining tools. But also for MRDM tools, the new representation may be attractive. A simple propositional condition will require two or three multi-relational refinements. More importantly, these refinements will only take effect in unison, making the pattern hard to discover by a greedy algorithm (such as Safarii). On the other hand, many other information related to individual records, such as remaining attributes or associated tables, will be lost in the transformation. Furthermore, any condition on the value of z regardless of y will become impossible as these values no longer appear in the same column.

Example 11.3 Consider the retail database described in Section 11.2 and assume the retailer in question offers a moderate selection of products. Furthermore, let us assume that the **item** table contains a quantity attribute, and therefore products appear only once in each basket. *basket_id* and *product_id* in the **item** table are not unique, but the combination is. Although the number of items will be a lot smaller than the product of the number of products and baskets (not every product is bought by every customer), we still accept a reverse pivot as a promising transformation. The

resulting table contains *basket_id*, as well as attributes for each product. Each field represents the quantity of a product in a particular basket. Although the other tables are still available, we could make the new table the target and analyse it with a propositional algorithm, e.g. Apriori [5]. The new table does not however allow us to work in multi-relational style, and use background knowledge from **product** and **class**, or query the quantity of items regardless of their product code. Clearly, the resulting table only serves a special purpose, that is being the input to limited algorithms such as Apriori.

11.4.4 Aggregate

The *aggregate* transformation involves the aggregation step, as described in Chapter 7. This transformation enriches the information in one table with aggregated information from another table, joined by a one-to-many association. This second table is left unaltered. The rule applies to two tables *A* and *B* under the following conditions:

- there is an association between *A* and *B* with multiplicity 1 on the *A* side, and 0..n or 1..n on the *B* side.
- There is at least one non-key attribute in *B*.

As was demonstrated in Chapter 8 and 9, aggregate functions can be a very powerful tool to capture local structure. Because the aggregate transformation preserves the original table, it improves the information available to MRDM algorithms. As Chapter 8 demonstrates, repeated aggregation may produce a flat table fit for analysis using traditional tools. The only downside to this transformation is the increased space required.

Example 11.4 Figure 11.5 shows the result of four aggregate rules applied to the Mutagenesis database. Information from **atom** and **bond** is aggregated at **molecule**, and **bond** is aggregated twice at **atom**, once for each association. The appropriate set of aggregate functions depends on the data type of the attributes.

11.4.5 Select Attributes

The *select attributes* transformation performs an obvious, yet practical and common operation: produce a table that contains a subset of the attributes of the original table. The rule applies to any table that has one or more non-key attributes. Clearly this rule depends entirely on input from the user.

11.4.6 Select tables

The *select tables* transformation provides a basic service: it removes tables from the data model (but not from the database) as indicated by the user.

11.4.7 Create Indexes

The *create indexes* transformation does not change anything in the data model or in the underlying data, and is therefore not strictly a pre-processing operation. It is however, a very useful operation before any MRDM run, as it will improve the performance of the RDBMS on data mining primitives. The transformation will create database indexes for the primary and foreign key related to each association in the data model. The data structure that each index represents will speed up any joins performed over the association. The tedious task of considering the structure of the database, and subsequently creating the necessary indexes is thus automated by ProSafarii.

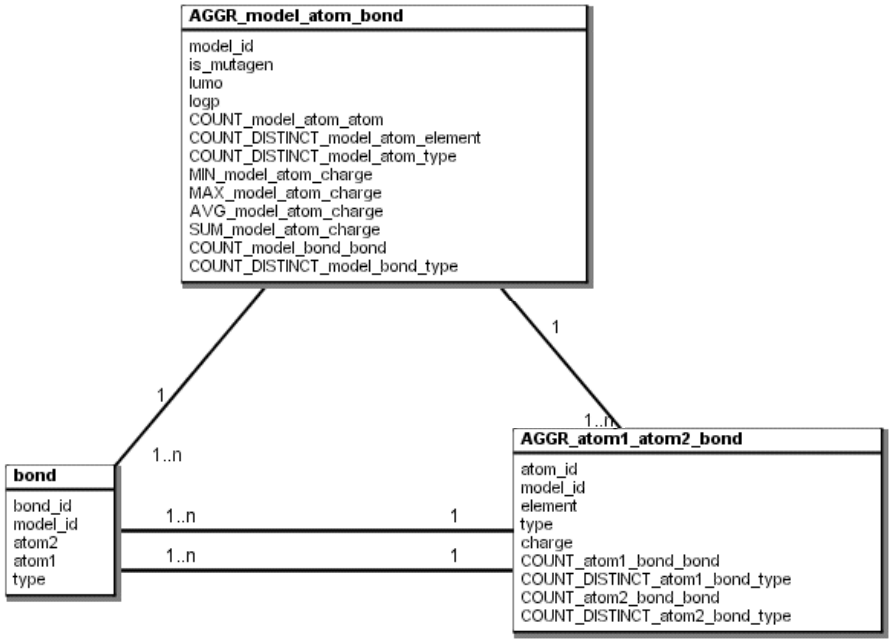


Figure 11.5 Two aggregate rules applied to the Mutagenesis database.

This page intentionally left blank

12 Conclusion

12.1 Contributions

In this section we outline the main contributions of this thesis.

We have defined Structured Data Mining as the genus of Data Mining paradigms concerned with structured data. SDM provides a unified framework for important concepts shared by the different paradigms. We have shown how, and to what extent four popular SDM approaches implement these concepts, and in what areas each of these paradigms excels. Specifically, we have outlined Multi-Relational Data Mining, a paradigm that is concerned with structured data in relational form.

We have demonstrated how structured individuals can be described by features based on local structure. Such features fall into two classes: *existential features*, which express the presence of specific substructures, and *aggregate functions*, which express global properties of groups of parts. Three pattern languages (with increasing expressive power) have been defined that can be used to select individuals, or define subgroups. *Selection graphs* can be used to select individuals purely on the basis of existential features. *Extended selection graphs* also work with existential features, but add the possibility of complementary patterns, a necessity for Divide-and-Conquer approaches. Finally, *generalised selection graphs* offer the possibility of aggregate functions, combined with existential features.

An important aspect of our MRDM framework is declarative bias. Relational databases are generally well-designed, and the underlying data model is an important means of guiding the search for patterns and achieving efficiency. We have introduced the popular Unified Modeling Language (UML) as declarative bias language in MRDM. The UML data model that describes the structure of the database both acts as a restriction on the shape of individuals, as well as on that of tentative models of the data. For Data Mining software to work with the graphical data models in UML, we have defined an XML dialect called MRML (Multi-Relational Modelling Language) that represents the same information serially.

In order to support the analysis of industrial-size databases stored in an RDBMS, we have outlined a Client/Server architecture that organises the major computational processes in data analysis. This

architecture revolves around a set of multi-relational data mining primitives that each support one of the refinement steps within the different pattern languages. The primitives are a means of separating the data access challenge from the actual pattern discovery. The information contained in such a primitive is both sufficient for the algorithm to guide the search, as well as (relatively) easy to compute for the query processing component of the architecture (typically the RDBMS).
We have defined four algorithms for analysing multi-relational data. Three of these algorithms work with one of the above-mentioned pattern languages in turn. A fourth algorithm involves pre-processing of the data in order to arrive at a representation that is suitable for existing propositional algorithms. A comparison of these four algorithms is given in Section 12.3. The Multi-Relational Rule Discovery algorithm that works with selection graphs has been the basis of a commercial MRDM tool, called Safarii. This systems is based on the presented architecture.
The MRDM technology presented in this thesis mainly pertains to the modelling step of a larger process, which is often referred to as Knowledge Discovery in Databases (KDD) [33]. As we recognise the importance of the remaining steps in this process, we have presented an MRDM project blueprint. This blueprint outlines how we can arrive at a dataset that is suitable for mining. The blueprint focuses on two major tasks, Data Understanding and Data Preparation. The first task is concerned with deciding the best way to approach an analysis task on the basis of available meta-data describing the database. The second task deals with pre-processing the data in a way that emphasises promising features of the individuals, and that is optimal for the mining algorithm of choice.
In order to support the data analyst in these activities, we have developed an MRDM pre-processing consultant. The application, called ProSafarii, makes a superficial scan of the data model and the actual database, and suggests a number of interesting transformations of the database. Each of these transformations potentially allows a number of further transformations. After validation by the user, the system performs selected transformations by executing the necessary queries in the RDBMS. It furthermore updates the data model to enable direct analysis of the derived database by MRDM tools such as Safarii. As a by-product of the transformations that are found to be applicable, the system provides a basic degree of insight into the structure of the data model.

12.2 Validity of MRDM Approach

This thesis deals with Multi-Relational Data Mining. From the start we have defined our MRDM approach as one of a collection of Structured Data Mining paradigms. Although based on different representational settings, these paradigms are very similar in their treatment of structured data. So has our emphasis on MRDM been justified, or have we just reformulated achievements of SDM paradigms with a longer history, such as ILP?
One of our motivations for considering MRDM in detail was to benefit from a large body of research in the field of relational databases, for example where data modelling, query languages and query optimisation is concerned. The intended ‘paradigm shift’ would inspire new developments in SDM that seem logical steps in a multi-relational setting, but may have been overlooked in alternative paradigms. A certain degree of reinventing the wheel is a necessary element of our whish to take a fresh look at the analysis of structured data. Still, we can outline a number of important developments that validate MRDM as a strong competitor to alternative SDM paradigms. The validity of MRDM is further supported by empirical results as described in the different sections on experimental evaluation. We mention a number of areas where MRDM outperforms related paradigms.
First, there is the strong role of the data model of the database as declarative bias for the search process. Rather than trying to define the shape of patterns the algorithm is required to discover, MRDM uses the structure of the database in UML to guide the search. The same data model is used

in transformation-based approaches such as propositionalisation or semi-automated pre-processing (see Section 11.3). A similar database orientation inspires a further strong point of MRDM: hypothesis testing in the RDBMS. The Client/Server architecture that separates the mining algorithms from the data handling offers a range of opportunities for query optimisations (e.g. parallelisation, batch processing) independent of the actual mining algorithm. The proposed Data Mining primitives already achieve an initial level of optimisation by combining several hypotheses into one query.
A further characteristic of MRDM is its incorporation of aggregate functions in pattern languages. Different experiments show that such features can be very powerful, particularly in the presence of numeric data and high levels of non-determinacy. Although existential features are a logical choice in certain SDM paradigms, for example first-order logic in ILP or subgraphs in GM, they are simply too restrictive in a range of application domains.
Although not entirely exclusive to MRDM, the full treatment of numeric data seems to be typical for this paradigm. This subject has often been ignored because it conflicts with a number of popular concepts, such as exhaustive search or frequent pattern discovery. Still, empirical comparisons show that SDM algorithms benefit from good support for numeric data. See for example Chapter 6, or [69, 85].
These qualities of MRDM clearly show that our emphasis on this field has paid off, and in fact MRDM deserves further attention in the future. However, the choice of paradigm to employ when dealing with structured data will in practice be a question of taste. Because the different paradigms have a lot in common, people are likely to base their choice on 'superficial' properties, such as those outlined in Table 2.1.

12.3 Overview of Algorithms

In this thesis we have presented four algorithms that approach the analysis of multi-relational data in a variety of ways. Although it is often wise to approach a problem from multiple sides, and try more than one algorithm, one could wonder which of these four algorithms is the preferred choice for a given MRDM problem. In this section we will shed some light on the particulars of each algorithm.
The most important feature of a problem that influences the choice of algorithm is whether the task is *predictive* or *descriptive* in nature. If the task is predictive, one is interested in global models that give a unique classification of each individual, be it an individual that occurs in the database at hand or in as yet unseen databases. Predictive models are primarily aimed at classifying unseen individuals for which the target value is unknown. A descriptive data Mining task on the other hand is concerned with producing local models that explain particular aspects of the data, or uncover partial dependencies. As such, the aim is possibly fragmented but increased understanding, rather than a full explanation that is geared towards prediction. The algorithms SGRules and GSGRules are descriptive. MRDTL (as an instance of our decision tree induction framework) and RollUp + C5.0 (as an instance of Polka) are predictive. Note that the predictive nature of this last algorithm is the result of using C5.0. RollUp can be employed in a descriptive fashion, given the right propositional algorithm.
RollUp and GSGRules both include aggregate functions. As we have seen, aggregate functions are a potentially richer means of capturing local structure compared to existential features only. Section 9.3 demonstrates in practice the increased descriptive power of GSGRules compared to SGRules. However, this power comes at the price of lower efficiency or a smaller proportion of the search space being considered. Furthermore, patterns that include aggregate functions can be more difficult to interpret.

As a further means of comparing the algorithms, we can consider the three complexity measures for multi-relational patterns. Depending on the complexity of the data model, or of patterns as they can be expected prior to the analysis, different algorithms may be suitable. A high refinement-depth is shared by all algorithms as this measure makes no distinction between types of refinements (e.g. propositional, existential, aggregation). RollUp + C5.0 is the champion of refinement-depth, as it consists of two steps (see Section 8.4.2).
The aggregate function-based algorithms have a moderate association-depth. Introducing multiple levels of aggregation seems to have its limits, even if *exists* (or a simple join in RollUp) is a common aggregate function. In theory, SGRules and MRDTL have a high association-depth. In practice however it turns out that SGRules tends to spend its computational resources searching the breadth of the search space, at the cost of the association-depth. Given a data model that contains little 'distraction' in terms of (numeric) attributes and, on the other hand, the opportunity for chains of association refinements, SGRules will search association-deep. MRDTL will look association-deep as a result of its efficient structure.
RollUp + C5.0 and GSGRules have a low association-width because of our choice of aggregate function classes. Although GSGRules does allow further conditioning within aggregated groups of records, this tends to be rare in databases we have considered. This is probably due to the way numeric thresholds for aggregate conditions are selected on the whole group. These thresholds may be less suitable once the aggregated group changes.
Table 12.1 recapitulates these characteristics.

	task	aggregate functions	speed	refinement-depth	association-depth	association-width
SGRules	descriptive	–	medium	high	medium	medium
MRDTL	predictive	–	high	high	high	medium
RollUp+C5.0	predictive	+	medium	very high	medium	low
GSGRules	descriptive	+	low	high	medium	low

Table 12.1 Characteristics of the four MRDM techniques.

12.4 Conclusion

In this thesis we have demonstrated that Multi-Relational Data Mining is inherently more powerful than Propositional Data Mining. There clearly is a large class of Data Mining problems that cannot be successfully approached using a single table as representational setting. These problems, which can be characterised by the presence of internal structure within the individuals they deal with, can successfully be approached by the multi-relational tools and techniques that are the subject of this thesis.
MRDM techniques are not the only ones that deal with structured data. We have presented a genus of Structured Data Mining paradigms that each approach the representation of data, and consequently the manipulation and analysis of the database, from a unique 'tradition'. As we have argued in Section 12.2, MRDM is an important member of this family of paradigms. Its particular strength lies in how it employs a number of concepts from relational database theory to capture more complex characteristics of the data, and achieve efficiency and scalability. Additionally, the dominance in industry of its underlying relational tradition makes MRDM an obvious choice.
Although our main emphasis has been on MRDM, we recognise the value of approaching problems in the more abstract setting of Structured Data Mining. By combining achievements that have been made relatively independently of one another, a richer set of techniques becomes available, and redundant development can be prevented. Furthermore a unified approach aids the comparison of existing techniques, which are mainly representation-specific. We therefore see the generalisation

of techniques from the individual paradigms, and integration of common ideas in SDM, as an important direction for future research.
In the remainder of this chapter, we consider some topics that are natural and promising extensions of the research described in this thesis.

12.5 Pattern Languages

In this thesis we have introduced a number of pattern languages of increasing expressive power. The most basic language SG is very similar in power to pattern languages that are employed in other Structured Data Mining paradigms, such as ILP or Graph Mining. These kinds of language, which rely heavily on existential features, are often used as a result of the traditional role of such languages in the knowledge representation scheme that inspired the approach (e.g. Prolog in ILP). The promising results obtained by using GSG suggests that existential feature-based SMD has its limitations, and that considering more powerful pattern languages can have a positive effect on the quality of the models produced. With this positive outcome in mind, one could wonder whether further extensions of the pattern languages used might be beneficial. Such extensions will fall into one or more of the following categories:

- Syntax. Which patterns are well-formed in the extended pattern language?
- Semantics. When are individuals covered by a given pattern?
- Refinement operator. What subset of the well-formed patterns is derivable through repeated refinement?

Below we give some suggestions for extensions to the introduced pattern languages, along with the required changes in terms of syntax, semantics or refinement operator. The diagram in Figure 12.1 demonstrates how the different languages relate to each other. The ovals indicate languages that are syntactically equivalent.

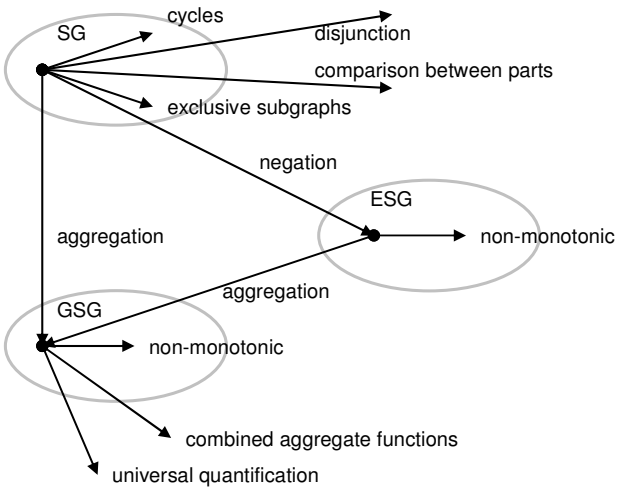


Figure 12.1 Possible pattern languages.

- Allow cycles in the selection graphs. This can be achieved by a simple addition to the normal refinement operator that adds edges between two existing nodes in the graph. The following pattern is an example:

  'the set of molecules that contain two atoms joined by a bond'

- Interpret subgraphs of a particular node connected by the same association as referring to distinct parts in the individual. Such an extension deals with the semantics of SG. Selection graphs that appear as usual now select different subgroups.

  'the set of molecules that contain a C-atom and a different atom of charge greater than $6.60 \cdot 10^{-7}$'

- Allow disjunctive conditions in selection nodes. The current set of conditions associated with each node represents a conjunction of conditions, whereas both the syntax and the semantics could be changed in order to incorporate both. Note that disjunction will introduce non-monotonic refinements.

  'the set of molecules that contain a Br, Cl, F, or I-atom'

  This example covers 22 individuals in the Mutagenesis database and has a novelty of –0.030 (see also Section 5.2).

- Allow comparison between selection nodes of the same class, for example, between two instances of the same attribute. Previously, comparison between parts was exclusive to the concept of associations. A less restrictive notion can be used to allow the comparison of attributes of parts other than the primary and foreign keys.

  'the set of molecules that contain 2 atoms of equal element'

- Include non-monotonic refinements in ESG. Adding conditions to *absent* subgraphs introduces a generalisation and is thus not acceptable in top-down algorithms such as presented in Chapter 6. Interesting patterns can however be considered if non-monotonic search is allowed.

  'the set of molecules that do not contain an O-atom'

- Include non-monotonic refinements in GSG.

  'the set of molecules that contain at most 5 C-atoms'

- Include universal quantification over associations: every part that is introduced over an association should meet the specified conditions. Because universal quantification captures local structure, one can think of it as a special type of aggregate condition, although the interaction between aggregate function and further conditions is different. Conditions on the subgraphs should not be interpreted as selections on the sets of parts that are aggregated, but rather as necessary features of all parts.

'the set of molecules that contain just C-atoms'

- Include nested aggregate functions that work over multiple associations. The virtual attribute created by the first aggregate function is the argument of a second function, and so on. This kind of nested aggregate function is produced by the pre-processing procedure Roll-Up, but a more general dynamic algorithm based on extensions of GSG does not exist.

  'the set of molecules that have an average bond-count per atom greater than 2.6'

- Include aggregate functions over a single association that are arithmetic functions of the basic set of aggregate function classes used thus far (count, min, max, etc.).

  'the set of molecules that have a difference between the minimum and maximum atom-charge less than $4.37 \cdot 10^{-5}$'

The list of flavours of the features to extract from structured data is long, and seemingly tempting. The expressive power of each of these extensions may be of benefit to upgraded Data Mining algorithms. Even if technical difficulties to implement these algorithms can be overcome, the increased expressiveness comes at a substantial price. One could wonder how transparent or understandable less trivial variations of the example patterns mentioned above are to human users. Also, the new constructs will increase the level of redundancy in the output (as well as in candidate queues), and leave room for many logically equivalent or at least very similar rules.
A more important consideration is the effect of such extensions on the size of the hypothesis space. Practical and user-friendly Data Mining systems will be allowed a fixed amount of time or number of hypotheses to test. Any increase in size of the hypothesis space will hence result in a smaller fraction actually being considered. As a consequence, good patterns may be overlooked. Although a bigger and richer hypothesis space holds the promise of better patterns, the chances of actually discovering such patterns will decrease. In our view, we have already reached the limit in this respect with the current implementation of rule induction in GSG.
Still, a number of the suggested patterns appear as natural as the kind of patterns that is considered by current MDRM algorithms. Specifically, the patterns that cannot be reached by top-down search deserve attention, albeit maybe in a restricted form. Before richer pattern languages can be considered however, the fairly straightforward search strategies presented in the previous chapters need to be refined, in order to explore the larger search space effectively. In the following section, we consider a number of necessary improvements in multi-relational search.

12.6 Improved Search

The search strategies employed in the Rule Discovery algorithms in this thesis are fairly straightforward. As a result, these algorithms may suffer from some effects that will become more important with enrichments of the pattern languages as suggested in the previous section. In order to continue to be able to effectively and efficiently consider a promising subset of hypotheses, these undesirable issues must be addressed.
The most obvious problem is poor efficiency due to a large number of hypotheses being tested on the database. An important reduction in the data mining primitives executed can be achieved by removing redundant hypotheses. The current refinement operators (as defined in Chapter 4, 6 and 9) will sometimes produce the same selection graph multiple times as a result of alternative orderings of refinements. The solution is to use an *optimal* refinement operator: every point in the search space can only be reached along a single path of refinements. This optimality is typically achieved

by defining an ordering on refinements through lexicographic comparison. Unfortunately, this syntactic notion of optimality only works in combination with exhaustive search. It conflicts with the heuristic search strategies that are vital in larger industrial applications. Different classes of heuristics may choose refinement paths that are prohibited by the optimal refinement operator.

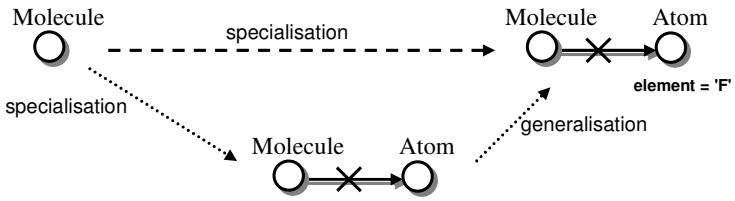
A more pragmatic approach to avoiding redundant hypothesis testing is to maintain a data structure of previously considered patterns, and check for redundancy before testing on the database. This method has attained some popularity in Propositional Data Mining due to the availability of efficient indexing structures for retrieving equivalent or otherwise relevant previous hypotheses. The trade-off between executing a database query and searching in the data structure has thus been in favour of the latter. In Structured Data Mining, such indexing of patterns is less trivial and more research is required to produce data structures that justify the effort.

Another important method of reducing hypothesis testing is known as *optimistic estimate pruning*. This method is applied for example in the ILP-systems MIDOS [31, 107] and Tertius [36]. It relies on the idea that it is possible to estimate the score of the best pattern that can ever be derived from the pattern at hand, keeping in mind that there is a limit to the amount of change in distribution in the subgroups resulting from the minimum coverage constraint. If this optimistic estimate is worse than the score of the current pattern or its ancestors, or worse than the minimum score, then the entire subtree of the search space can be pruned. Optimistic estimates can be derived for all of the popular rule measures. These estimate functions typically consider the 'ideal' pattern, for example one that covers all the positive examples, and compute its score.

The important challenge in richer pattern languages deals with non-monotonic refinement operators. Such operators are refinement operators that are not specialisation operators: refined patterns do not necessarily produce subsets of the original subgroup. This property makes top-down search impossible. In itself this need not be a problem. Exhaustive search of a particular hypothesis space defined by a maximum pattern complexity (i.e. refinement depth) can be done in any order. In practical cases however, exhaustive search is unrealistic, and some form of heuristic (parallel) hill-climbing is required that relies heavily on monotonicity. So what are the effects of dispensing with monotonicity? First of all, there is no pruning on minimum coverage. Although we can filter the ultimate result on coverage, intermediate hypotheses along the refinement path may even have zero coverage. An important source of efficiency is thus lost. Important Data Mining algorithms such as Apriori [5] rely on minimum coverage pruning for their practical applicability. Secondly, the optimistic estimate pruning mentioned above cannot be applied as it relies both on a minimum coverage and on derived patterns being a subset of the current pattern. Finally, the heuristics used to guide the search (typically the score of the pattern at hand, or an optimistic estimate of its subtree) become problematic.

The solution is to use a form of look-ahead in order to bridge intermediate hypotheses with a low coverage. A superficial form of such a look-ahead involves deriving a monotonic refinement operator from a non-monotonic one by combining refinement steps. This can be achieved by preceding every generalisation by a specialisation, such that the combined result is a specialisation. This combination of refinement steps is not unlike the look-ahead in multi-relational decision tree induction. Unfortunately, this solution is limited as is excludes refinement paths that involve multiple consecutive generalisations.

Example 12.1 Consider the following refinements on extended selection graphs in the Mutagenesis domain. Although there is a generalisation involved, the two steps still constitute a specialisation.



A more general solution entails look-ahead over multiple refinements. If exhaustive search needs to be prevented, this can only be achieved by a heuristic that estimates how promising the pattern at hand is, given a limit on the refinement depth. Rather than working with statistical clues in the traditional way, such a heuristic could also involve syntactic properties of the pattern. Syntactic constructs that seem restrictive, but that are known to enable generalisation refinements will be favoured. The effects of overly specific refinements is thus balanced by the promise of future refinements that produce a more desired coverage. The search will still be non-monotonic, but properly directed. Clearly, the challenge is to find a proper weighting system between statistical and syntactic values in the heuristic.

This page intentionally left blank

Appendix A: MRML

The Document Type Definition (DTD) of the MRML file format is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mrml (version, database, datamodel)>
<!ELEMENT database (name, server?, host?, user?, password?)>
<!ELEMENT datamodel (name, table+, association*)>

<!ELEMENT table (name, attribute+)>
<!ELEMENT attribute (name)>
<!ATTLIST attribute type (nominal|numeric|key|primarykey) #REQUIRED>
<!ATTLIST attribute id ID #IMPLIED>

<!ELEMENT association (name, keyref, keyref)>
<!ATTLIST association direction (none|forward|reverse|both) "forward">
<!ELEMENT keyref EMPTY>
<!ATTLIST keyref id IDREF #REQUIRED>
<!ATTLIST keyref multiplicity (zeroorone|one|zeroormore|oneormore) "zeroormore">

<!ELEMENT version (#PCDATA)>
<!ELEMENT server (#PCDATA)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

An MRML file describes in detail the data model of a database. Let us consider the major components in turn.

```
<!ELEMENT database (name, server?, host?, user?, password?)>
```

This rule sates that the `database` element must contain a name of a physical database, and optionally information concerning the location and access to the database.

```
<!ELEMENT datamodel (name, table+, association*)>
```

This states that a data model consists of at least one table, and zero or more associations.

```
<!ELEMENT table (name, attribute+)>
<!ELEMENT attribute (name)>
```

Every `table` element consists of at least one `attribute` that in turn consists of a `name` element.

```
<!ATTLIST attribute type (nominal|numeric|key|primarykey) #REQUIRED>
<!ATTLIST attribute id ID #IMPLIED>
```

The rules state that an `attribute` element contains one or two XML-attributes. An attribute at least contains a `type` element, which indicates the nature of the data. The values `nominal` and `numeric` refer to actual data, whereas `key` and `primarykey` indicate the use of this attribute in foreign key relations. The optional XML-attribute `id` is used to give a unique identifier for reference by other XML-elements.

```
<!ELEMENT association (name, keyref, keyref)>
<!ATTLIST association direction (none|forward|reverse|both) "forward">
```

An `association` element has a name and two references to keys. Furthermore the `association` has an XML-attribute `direction`, which specifies in which direction an association may be traversed.

```
<!ELEMENT keyref EMPTY>
<!ATTLIST keyref id IDREF #REQUIRED>
<!ATTLIST keyref multiplicity (zeroorone|one|zeroormore|oneormore) "zeroormore">
```

The `keyref` element contains one mandatory XML-attribute `id` that states the identifier of a particular attribute in a table. An optional XML-attribute `multiplicity` fixes the multiplicity of the association in question (one on each side).

Bibliography

1. Abe, K., Kawasoe, S., Asai, T., Arimura, H., Arikawa, S. *Optimized Substructure Discovery for Semi-Structured Data*, In Proceedings of PKDD 2002, LNAI 2431, 2002
2. Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web*, Morgan Kaufmann, 2000
3. Adriaans, P. *Adaptive System Management*, in Advances in Applied Ergonomics, In Proceedings of ICAE'96, Istanbul, 1996
4. Adriaans, P., Zantinge, R. *Data mining.* Addison-Wesley, 1996
5. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. *Fast Discovery of Association Rules*, in Artificial Intelligence, 1997
6. Alphonse, É., Rouveirol, C. *Selective Propositionalization for Relational Learning*, In Proceedings of PKDD '99, LNAI 1704, 1999
7. Appice, A., Ceci, M., Malerba, D. *MR-SMOTI: A Data Mining System for Regression Tasks Tightly-Coupled with a Relational Database*, In Proceedings of KDID-2003, 2003
8. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., and Arikawa, S. *Efficient substructure discovery from large semi-structured data*, In Proceedings of SDM2002, 2002
9. Blockeel, H. *Top-Down Induction of First Order Logical Decision Trees*, Ph.D. thesis, 1998
10. Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H. *Executing Query Packs in ILP*, In Proceedings of ILP 2000, LNAI 1866, 2000
11. Blockeel, H., De Raedt, L. *Relational knowledge discovery in databases,* In Stephen Muggleton, editor, In Proceedings of ILP '96, LNAI 314, pp. 199-211, 1996
12. Blockeel, H., De Raedt, L. *Top-down induction of first order logical decision trees*, Artificial Intelligence, 101(1-2):285-297, June 1998
13. Blockeel, H., De Raedt, L., Jacobs, N., Demoen, B. *Scaling up inductive logic programming by learning from interpretations.* Data Mining and knowledge Discovery, 3(1):59-93, March 1999

14. Blockeel, H., De Raedt, L., Ramon, J. *Top-down induction of clustering trees,* In Proceedings of ICML'98, 55-63, 1998
15. Boncz, P. *Monet, a next-Generation DBMS Kernel for Query-Intensive Applications*, Ph.D. thesis, 2002
16. Braga, D., Campi, A., Klemettinen, M., Lanzi, P. *Mining Association Rules from XML Data*, In Proceedings of DaWak 2002, 2002
17. Bray, T. *Introduction to the Annotated XML Specification*, http://www.xml.com/axml/axml.html
18. Chapman, P., Clinton, J., Khabaza, T., Reinartz, T., Wirth, R. *The CRISP-DM Process Model*, 1999
19. Cook, D., Holder, L. *Graph-Based Data Mining*, Intelligent Systems & their Applications, 15(2):32-41, 2000
20. Dasu, T., Johnson, T. *Exploratory Data Mining and Data Cleaning*, Wiley, 2003
21. Date, C. *An Introduction to Database Systems, Volume I*, The Systems Programming Series, Addison-Wesley, 1986
22. Dehaspe, L. *Frequent Pattern Discovery in First-Order Logic*, Ph.D. thesis, 1998
23. Dehaspe, L., and De Raedt, L. *Mining association rules in multiple relations*, In Proceedings of ILP'97, LNAI 1297, 1997
24. Dehaspe, L., Toivonen, H., *Discovery of frequent Datalog patterns*, Data Mining and Knowledge Discovery, 3(1), 1999
25. Dehaspe, L., Toivonen, H., King, R. *Finding frequent substructures in chemical compounds*, In Proceedings of KDD '98, 30-36, 1998
26. Dewhurst, N., Lavington, S. *Knowledge Discovery from Client/Server Databases,* In Proceedings of PKDD '98, 1998
27. Dietterich, T., Lathrop, R., Lozano-Pérez, T. *Solving the multiple-instance problem with axis-parallel rectangles*, Artificial Intelligence, 89(1-2):31-71, 1997
28. Džeroski, S. *Inductive Logic Programming and Knowledge Discovery in Databases,* in [33], AAAI Press, 1996
29. Džeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., Van Laer, W., *Experiments in Predicting Biodegradability*, In Proceedings of ILP '99, LNAI 1634, 1999
30. Džeroski, S., Lavrač, N., *An Introduction to Inductive Logic Programming,* In [31]
31. Džeroski, S., Lavrač, N., *Relational Data Mining*, Springer-Verlag, 2001
32. *Extensible Markup Language (XML).* http://www.w3.org/XML/
33. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996
34. Finkelstein, S., Mattos, N. Mimick, I., Pirahesh, H., *Expressing recursive queries in SQL*, ISO WG3 report X3H2-96-075, March, 1996
35. Flach, P. *The geometry of roc space: Understanding machine learning metrics through roc isometrics*, In Proceedings of ICML 2003, pp. 194–201, 2003
36. Flach, P., Lachiche, N. *Confirmation-Guided Discovery of First-Order Rules with Tertius*. Machine Learning, 42(1/2):61-95, January 2001
37. Freitas, A., Lavington, S. *Using SQL-primitives and parallel DB servers to speed up knowledge discovery in large relational databases*, In Proceedings of EMCSR '96, 1996.
38. Friedman, N., Getoor, L., Koller, D., Pfeffer, A., *Learning Probabilistic Relational Models,* In Proceedings of IJCAI '99, 1999
39. Fürnkrantz, J. Flach, P. *ROC 'n' Rule Learning - Towards a Better Understanding of Covering Algorithms*, to appear in the Machine Learning journal
40. Giudici, P. *Applied Data Mining*, Wiley, 2003

41. *Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1*, http://www.w3.org/XML/1998/06/xmlspec-report.htm
42. Hahn, M. *Info Charger Data Sheet,* http://www.tektonic.de/ icdatash.htm, 1997
43. Hand, D., Mannila H., Smyth, P., *Principles of Data Mining*, MIT Press, 2001
44. Herlaar, L. *Diagnosing Performane of Relational Database Managament Systems*, technical report, Utrecht University, 1995
45. Holsheimer, M., Kersten, M., Mannila, H., Toivonen, H. *A Perspective on Databases and Data Mining*, In Proceedings of KDD '95, 1995
46. Holsheimer, M., Kersten, M., Siebes, A. *Data Surveyor: Searching the nuggets in parallel*, In [33]
47. Ibraheem, S., Kokar, M., Lewis, L. *Capturing a Qualitative Model of Network Performance and Predictive Behavior*, Journal of Network and System Management, vol 6, 2, 1997
48. Inokuchi, A., Washio, T. and Motoda, H. *An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data*, In Proceedings of PKDD 2000, LNAI 1910, 2000
49. Jensen, D., Neville, J., Hay, M. *Avoiding Bias when Aggregating Relational Data with Degree Disparity*, In Proceedings ICML-2003, 2003.
50. John, G.H., Lent, B. *SIPping from the Data Firehose*, In Proceedings of KDD '97, 1997
51. Klösgen, W. *Explora: A Multipattern and Multistrategy Discovery Assistant,* In [33]
52. Klösgen, W. *Deviation and Association Patterns for Subgroup Mining in Temporal, Spatial, and Textual Data Bases,* In Proceedings RSCTC'98, 1998
53. Klösgen, W., May, M. *Spatial Subgroup Mining Integrated in an Object-Relational Spatial Database*, In Proceedings of PKDD 2002, LNAI 2431, 2002
54. Knobbe, A. *Data Mining for Adaptive System Management,* in Proceedings of PADD '97
55. Knobbe, A. *Towards Scalable Industrial Implementations of ILP,* ILPNet2 Seminar on ILP & KDD, Caminha, Portugal, 1998
56. Knobbe, A., Adriaans, P. *Analysing Binary Associations*, in Proceedings of KDD '96
57. Knobbe A., Adriaans, P. *Discovering Foreign Key Relations in Relational Databases*, In Proceedings of EMCSR '96, 1996
58. Knobbe, A., Blockeel, H., Siebes, A. Van der Wallen, D. *Multi-Relational Data Mining,* In Proceedings of Benelearn '99, 1999
59. Knobbe A., De Haas, M., Siebes, A. *Propositionalisation and Aggregates,* In Proceedings of PKDD 2001, LNAI 2168, pp. 277-288, 2001
60. Knobbe, A., Den Heijer, E., Waldron, R. *A Practical View on Data Mining*, in Proceedings of PAKM '96
61. Knobbe, A., Siebes, A., Blockeel, H., Van der Wallen, D. *Multi-Relational Data Mining, using UML for ILP*, In Proceedings of PKDD 2000, LNAI 1910, 2000
62. Knobbe, A., Siebes A, Van der Wallen, D. *Multi-Relational Decision Tree Induction*, In Proceedings of PKDD'99, LNAI 1704, pp. 378-383, 1999
63. Knobbe, A., Siebes, A., Marseille, B. *Involving Aggregate Functions in Multi-Relational Search*, In Proceedings of PKDD 2002, LNAI 2431, 2002
64. Knobbe, A. *A Guide to MRML,* http://www.kiminkii.com/ mrml.html
65. Kramer, S., Pfahringer, B., Helma, C. *Stochastic Propositionalization of non-determinate background knowledge*, In Proceedings of ILP '98, 1998
66. Kramer, S. *Structural regression trees,* In Proceedings of AAAI '96 , 1996
67. Kramer, S. *Relational Learning vs. Propositionalization*, Ph.D thesis, 1999
68. Krogel, M., Wrobel, S. *Transformation-Based Learning Using Multirelational*

Aggregation, In Proceedings of ILP 2001, LNAI 2157, pp. 142-155. 2001
69. Krogel, M., Rawles, S., Železný, F., Flach, P., Lavrač, N., Wrobel, S. *Comparative Evaluation of Approaches to Propositionalization*, In Proceedings of ILP 2003, 2003
70. Lavrač, N., Dzeroski, S. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994
71. Lavrač, N., Džeroski S. *Inductive Logic Programming,* In Proceedings of ILP-97, 1997
72. Lavrač, N., Džeroski, S., Grobelnik, M. *Learning nonrecursive definitions of relations with LINUS*, In Proceedings of EWSL'91, 1991
73. Lavrač, N., Flach, P., Zupan, B. *Rule Evaluation Measures: A Unifying View*, In Proceedings of ILP '99, LNAI 1634, 1999
74. Leiva, H., Atramentov, A., Honavar, V. *Experiments with MRDTL – A Multi-relational Decision Tree Learning Algorithm*, In Proceedings of Workshop MRDM 2002, 2002
75. Lindner, G., Morik, K. *Coupling a relational learning algorithm with a database system,* Workshop Notes of the MLNet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, 1995
76. Manegold, S., Pellenkoft, A., Kersten, M. *A Multi-Query Optimizer for Monet*, In Proceedings of BNCOD 2000, LNCS 1832, 2000
77. Mannila, H., Toivonen, H. *On an algorithm for finding all interesting sentences*, In Proceedings of EMCSR '96, 973-978, 1996
78. Martin, L., Moal, F., Vrain, C. *A Relational Data Mining Tool Based on Genetic Programming,* In Proceedings of PKDD '98, Nantes, France, 130-138, 1998
79. Matsuda, T., Horiuchi, T., Motoda, H., Washio, T., et al. *Graph-bases induction for general graph structured data*, In Proceedings of DS'99, 340-342, 1999
80. Miyahara, T., Shoudai, T., Uchida, T., Kuboyama, T., Takahashi, K., Ueda, H. *Discovering New Knowledge from Graph Data Using Inductive Logic Programming*, In Proceedings of ILP '99, LNAI 1634, 1999
81. Miyahara, T., Shoudai, T., Uchida, T., Takahashi, K. and Ueda, H. *Discovery of frequent tree structured patterns in semistructured web documents*. In Proceedings PAKDD2001, 47-52, 2001
82. Morik, K., Brockhausen, P., *A Multistrategy Approach to Relational Knowledge Discovery in Databases,* in Machine Learning 27(3), 287-312, Kluwer, 1997
83. Muggleton, S. *Inverse entailment and Progol.* New Generation Computing, 13:245-286, 1995
84. Neville, J., Jensen, D., Friedland, L., Hay, M. *Learning Relational Probability Trees*, In Proceedings of SIGGKDD '03, 2003
85. Perlich, C., Provost, F. *Aggregation-Based Feature Invention and Relational Concept Classes*, In Proceedings of SIGGKDD '03, 2003
86. Provost, F., Fawcett, T. *Analysis and visualization of classifier performance: comparison under imprecise class and cost distribution.* In Proceedings KDD '97, 1998.
87. Pyle, D. *Data Preparation For Data Mining*, Morgan Kaufmann, 1999
88. Quinlan, R. *C4.5: Programs for Machine Learning,* Morgan Kaufmann, 1993
89. de Raedt, L. (Ed.) *Advances in Inductive Logic Programming*, IOS Press, 1996
90. Rang, H., Dale, M. *Pharmacology*, Churchill Livingstone Inc., Longman Group UK Limited, 1993
91. Ribeiro, J., Kaufman, K., and Kerschberg, L. *Knowledge discovery from multiple databases,* In Proceedings of KDD '95, 240-245, 1995
92. Rumbaugh, J., Booch, G., Jacobson, I. *Unified Modeling Language Reference Manual*, Addison Wesley, 1998
93. Safarii, http://www.kiminkii.com/safarii.html

94. Si Alhir, S. *UML in a Nutshell*, O'Reilly & Associates, 1998
95. Srinivasan, A., King, R., *Feature construction with ILP: a study of quantitative predictions of biological activity by structural attributes*, In Proceedings of ILP '96, 1996
96. Srinivasan, A., Muggleton, S., Sternberg, M., King, R. *Theories for mutagenicity: A study in first-order and feature-based induction*, Artificial Intelligence, 85(1,2), 1996
97. Srinivasan, A., King, R., Bristol, D., *An Assessment of ILP-Assisted Models for Toxicology and the PTE-3 Experiment*, In Proceedings of ILP '99, LNAI 1634, 1999
98. Todorovski, L., Džeroski, S. *Experiments in Meta-level Learning with ILP*, In Proceedings of PKDD '99, LNAI 1704, 1999
99. Toivonen, H. *Discovery of Frequent Patterns in Large Data Collections,* Ph.D. thesis, 1996
100. Ullman, I.D. *Principles of Databases and Knowledge-Based Systems,* Volume I, Computer Science Press, 1988
101. Ullman, J., Widom, J., *A First Course in Database Systems*, Prentice Hall, 2001
102. Van Laer, W. *From Propositional to First Order Logic in Machine Learning and Data Mining*, Ph.D. thesis, 2002
103. Van Laer, W., De Raedt, L., *How to Upgrade Propositional Learners to First Order Logic: A Case Study*, In [33]
104. Wang, K., Liu, H. *Discovering structural association of semistructured data*, IEEE Trans. Knowledge and Data Engineering (TKDE2000), 12(3):353-371, 2000
105. Watanabe, L., Rendell, L. *Learning structural decision trees from examples,* In Proceedings of IJCAI '91, 770-776, 1991
106. Workshop notes on Discovery Challenge PKDD '99, 1999
107. Wrobel, S. *An algorithm for multi-relational discovery of subgroups,* In Proceedings of PKDD '97, 78-87, 1997
108. *XML Path Language (XPath)*, http://www.w3.org/TR/xpath
109. *XQuery 1.0: An XML Query Language*, http://www.w3.org/TR/xquery
110. Yao, I., Lin, H. *Searching Multiple Databases for Interesting Complexes,* In Proceedings of PAKDD '97, 198-210, 1997
111. Zantinge, R., Adriaans, P. *Managing Client/Server.* Addison-Wesley, 1996

This page intentionally left blank

Index