A NEW HYBRID MULTI-RELATIONAL DATA MINING TECHNIQUE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEDA DAĞLAR TOPRAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

MAY 2005

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. İ. Hakkı Toroslu
Supervisor

Examining Committee Members

Prof. Dr. Adnan Yazıcı (METU, CENG) _____

Assoc. Prof. Dr. İ. Hakkı Toroslu (METU, CENG) _____

Asst. Prof. Dr. İlyas Çiçekli (Bilkent University) _____

Assoc. Prof. Dr. Nihan K. Çiçekli (METU, CENG) _____

Dr. Pınar Karagöz Şenkul (METU, CENG) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name:

Signature :

ABSTRACT

A NEW HYBRID MULTI-RELATIONAL DATA MINING TECHNIQUE

Toprak, Seda Dağlar

M.Sc., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. İ. Hakkı Toroslu

May 2005, 72 pages

Multi-relational learning has become popular due to the limitations of propositional problem definition in structured domains and the tendency of storing data in relational databases. As patterns involve multiple relations, the search space of possible hypotheses becomes intractably complex. Many relational knowledge discovery systems have been developed employing various search strategies, search heuristics and pattern language limitations in order to cope with the complexity of hypothesis space. In this work, we propose a relational concept learning technique, which adopts concept descriptions as associations between the concept and the preconditions to this concept and employs a relational upgrade of association rule mining search heuristic, APRIORI rule, to effectively prune the search space. The proposed system is a hybrid predictive inductive logic system, which utilizes inverse resolution for generalization of concept instances in the presence of background knowledge and refines these general patterns into frequent and strong concept definitions with a modified APRIORI-based specialization operator. Two versions of the system are tested for three real-world

learning problems: learning a linearly recursive relation, predicting carcinogenicity of molecules within Predictive Toxicology Evaluation(PTE) challenge and mesh design. Results of the experiments show that the proposed hybrid method is competitive with state-of-the-art systems.

Keywords: Multi-Relational Learning, ILP, Predictive Inductive Learning, Descriptive Inductive Learning, Inverse Resolution, Association Rule Mining, APRIORI, WARMR

ÖZ

YENİ BİR MELEZ ÇOK İLİŞKİLİ VERİ MADENCİLİĞİ TEKNİĞİ

Toprak, Seda Dağlar
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. İ. Hakki Toroslu

Mayıs 2005, 72 sayfa

Verileri ilişkisel veritabanlarında saklama eğilimi ve problem tanımlamalarını tek bir bağıntı ile yapmanın getirdiği sınırlamalar, çok ilişkili öğrenmeyi popüler hale getirmiştir. Bilgi örüntüleri birden fazla ilişki içermeye başladıkça, örüntü arama uzayı kolay işlenmeyecek kadar büyümüştür. Hipotez uzayının karmaşıklığı ile başa çıkmak için farklı arama stratejileri ve örüntü dil kısıtları kullanan birçok çok-ilişkili bilgi çıkaran sistem geliştirilmiştir. Bu çalışmada kavram öğrenme, kavram ile kavramı gerçekleme önkoşulları arasındaki eşleştirme olarak tanımlanmış ve ilişkisel kural madenciliği alanında buluşsal yöntem olarak kullanılan APRIORI kuralı örüntü uzayını küçültmek amacı ile kullanılmıştır. Önerilen sistem, kavram örneklerinden *ters çözünürlük* operatörü kullanılarak genel kavram tanımlarını oluşturan, ve bu genel örüntüleri APRIORI kuralını temel alan bir operatör yardımı ile özelleştirerek güçlü kavram tanımlamaları elde eden melez bir öğrenme sistemi olarak tanımlanabilir. Sistemin iki farklı versiyonu, üç popüler veri madenciliği problemi için test edilmiş ve sonuçlar önerilen sistemin, en gelişkin ilişkisel veri madenciliği sistemleri ile karşılaştırılabilir du-

rumda olduğunu göstermiştir.

Anahtar Kelimeler: İlişkisel Öğrenme, Tümevaran Mantıksal Programlama, Öngörülü Tümevarımsal Öğrenme, Tanımlayıcı Tümevarımsal Öğrenme, Ters Çözünürlük, İlişkisel Kural Madenciliği, APRIORI Algoritması, WARMR Sistemi

To Serkan *with love...*

ACKNOWLEDGMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. İ. Hakkı Toroslu, who encouraged and guided me in writing of this thesis. Thanks to the other committee members, Prof. Dr. Adnan Yazıcı, Assoc. Prof. Dr. İlyas Çiçekli, Assoc. Prof. Dr. Nihan K. Çiçekli and Dr. Pınar Karagöz Şenkul for their comments and suggestions.

I would like to express my deepest gratitude to my sweetheart, Serkan, who encouraged and supported me in this demanding study with his never ending patience.

I would also like to thank to my employer, Central Bank of the Republic of Turkey, for providing me time for studying whenever I needed.

Thanks are also to all of my friends for their direct or indirect help. Finally, my deepest thanks are to my parents who supported and motivated me with their never ending patience, tolerance and understanding throughout the study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Multi-Relational Learning

Initial knowledge acquisition systems have been developed to learn from propositional representation of problem domains. In propositional (attribute-value) learning, every target instance and the background knowledge related to that instance is represented by a single record in a table. This type of representation is infeasible to specify the relations between the subparts of the instance and one-to-many relations between the instance and its subparts. The inadequacy in representation results in incomplete learned concept descriptions.

Due to the impracticality of single-table data representation, multi-relational databases have become widespread in all computer-based processes. This has led to the need for multi-relational learning systems that directly apply to relational representations of structured problem domains. There are three key approaches in constructing relational learning systems: [1, 2].

- The system is composed of three parts: pre-processing, hypothesis construction and post-processing. In the preprocessing phase, the problem definition in relational form is transformed into propositional one. Then, one of the attribute-value learning systems, suitable for the data mining task, is applied. Finally, the induced if-then rules are transformed in re-

lational form. One of the ILP systems using this approach is the LINUS framework [3] that utilizes an embedded deductive hierarchical database (DHDB) interface in data transformation and one of three propositional learning systems among ASSISTANT [4], NEWGEM [5] and CN2 [6] according to the problem domain in induction phase. Due to the limitations of attribute-value representation mentioned, information loss is possible in transformation and propositional patterns are not as easily understandable as relational ones in a structured problem domain. Therefore, this method is not preferable.

- Attribute-value learning systems have been upgraded to the multi-relational counterparts in every branch of data-mining.

- New concept description systems have been introduced, in order to fulfill the task of defining unknown relations with the help of known background knowledge as logical programs.

Most relational upgrades of data mining systems and concept learning systems employ first-order predicate logic as representation language for background knowledge and data structures/patterns. The learning systems, which induce logical patterns or programs valid for given background knowledge, have been gathered under a research area, called Inductive Logic Programming (ILP), a subfield of Machine Learning and Logic Programming [7].

The propositional data structures used in data mining area (decision trees, if-then classification rules and association rules) have been extended to relational form in multi-relational data mining (MRDM) systems. Two most popular algorithms for inducing relational decision trees, SCART [8] and TILDE [9], are upgrades of the propositional decision tree induction systems, CART and C4.5, respectively. WARMR [10] upgrades the frequent item-set mining algorithm APRIORI for discovering relational frequent patterns and association rules. The key step of upgrading propositional distance-based algorithms is to redefine distance measure between structured objects. RIBL [11] defines a rela-

tional distance measure, and then adapts k-nearest neighbor approach to work on relational data. RDBC and FORC have utilized the RIBL distance measure; they adapt hierarchical agglomerative clustering and k-means approach to input relational data, respectively [12].

Concept learning aims at developing search techniques that efficiently traverse target concept description space consisting of logical Horn clauses. There are various approaches designed to solve this problem: [13]

- Top-down approach using information gain as search heuristics
- Top-down approach utilizing higher-order rule schemas to constrain search
- Bottom-up approach constraining search by generalizing from concept instances using inverse resolution operators
- Bottom-up approach avoiding search using relative least general generalization (RLGG) operator.

The first relational learning algorithm to use information gain based search heuristics was FOIL [14]. It uses an AQ-like covering approach [15] and it inherits the top-down search strategy from MIS [16], which is an early concept learning system. Recently, many systems that extend FOIL in various aspects have been introduced.

CIA [17], MODELER [18] and RDT [19] are among the methods that use higher-order rule schemas in order to guide search for learning logical clauses. CIA learns higher-order rule schemas from induced Horn clauses via substituting variables for both terms and predicates. The system employs these schemas in order to explain newly introduced concept instances. If there is no schema that may explain the new instance, the system introduces new rules via the rule learning system CLINT. MODELER accepts pre-defined higher-order rule schemas instead of learning them and put additional constraints in order to explain a concept instance as an instantiation of rule schemas. RDT utilizes the topology of clauses as an extra constraint for instantiating higher-order rules.

The search heuristics, information gain and higher-order rule schemas, have no proof-theoretic basis; therefore the search space of possible concept descriptions is not complete. The resolution rule that forms the basis of the logic programming paradigm is a sound and complete inference rule. Inverting this inference rule results in induction of refutation trees in a bottom-up fashion and systems employing inverse resolution operators have a proof-theoretic search strategy [13].

MARVIN [20] is the first ILP system inducing Horn clauses using an inverse resolution generalization operator. The hypothesis language of the system does not contain clauses with existential quantified variables and the system can not introduce new predicates. There is no search heuristics to direct the search; instead the oracle evaluates the quality of induced clauses.

CIGOL [21] employs three generalization operators based on inverse resolution, which are relational upgrades of absorption, intra-construction and truncation operators used in DUCE [22], whereas MARVIN utilizes only absorption operator. With these extra operators, CIGOL extends the learning capability of MARVIN with generating new predicate definitions. However, CIGOL also needs oracle knowledge to direct the induction process.

PROGOL [23, 24] is a bottom-up Horn clause induction system, that uses the inverse entailment operator in induction phase. In the system, after the positive instance is selected to be generalized, the most specific clause within the language constraints that entails the positive instance is constructed and the hypothesis space of clauses that are more general than this most specific clause is searched to find a qualified concept description. Rather than depending on the user's knowledge in induction step, it employs an evaluation measure of how well a clause explains all the examples with preference to shorter clauses.

GOLEM [25] is a bottom-up ILP that is based on the relative least general generalization operator.

1.2 Motivation

The motivation behind predictive ILP learning is to discover a complete and consistent hypothesis that best fits to the target concept instances. Each clause in the hypothesis represents a different structural pattern of the target concept. The number of positive instances fit to this structural pattern is the support of the concept clause. Predictive ILP systems do not utilize the "support" concept in pruning the search space; however descriptive ILP systems APRIORI and WARMR utilize the general support rule, the APRIORI trick, as a strong search heuristics. The need for the "support" concept in predictive ILP learning has led us to extend WARMR [10] query mining tool into a rule mining system that discovers frequent and confident relational rules, including linearly recursive clauses.

WARMR finds frequent relational queries employing a level-wise search strategy such that each frequent query is refined by adding one literal to the query at a time. This specialization operator results in a search space composed of disjoint sub-trees rooted at each frequent query. The possibility of generating recurrent candidate queries is high due to the search space structure. On the other hand, the chance of generating infrequent candidate clauses is also high since all the combinations of the added literal with the literals in the clause are needed to be frequent; therefore the system keeps track of infrequent clauses as a list, which increases the time complexity of the algorithm.

If the specialization operator joins two frequent queries that have all but one literal in common as in candidate generation step of APRIORI, the search lattice will be more compact and there will be no need for keeping a list of infrequent queries. In order to effectively prune the search space, the proposed concept learning system employs such an APRIORI-based refinement operator. Besides, our system utilizes the same search strategy and heuristics as in WARMR: bread-first search and the relational version of the APRIORI trick, explained in detail in Section 2.4, respectively.

In order to be able to apply APRIORI-like specialization operator, the first

level of the search lattice containing one-element item sets (horn clauses with one literal in the body) should be constructed beforehand. Pure top-down ILP systems waste time constructing clauses which cover no positive example, like WARMR. Therefore, the proposed system generalizes target concept instances in the presence of background knowledge as definite clauses with one literal in the body (two literal clauses) and populates first level of the search lattice with these generalizations so that each clause in the lattice covers at least one positive concept example. Absorption operator of inverse resolution introduced in [21], one of the most popular ILP generalization operators, is employed in this bottom-up induction step.

The main difficulty relational ILP systems face is searching in intractably large hypothesis spaces. The complexity of the hypothesis space stems from the variables in the hypothesis clauses, especially existential variables that occur in the body but not in the head of the clause. In order to cope with existential variables and learn efficiently, relational ILP systems put strong declarative biases on the semantics of hypotheses. GOLEM allows only *determinate* existence of literals (a determinate literal is a literal for which the values of output variables are uniquely determined for every possible set of input variable values) via ij-determinacy [25]. Besides, many multi relational rule induction systems, including GOLEM, PROGOL, WARMR and FOIL, require the user to determine the input-output modes of predicate arguments. Since mode declarations require an oracle level Prolog and domain knowledge, it is risky to expect such a declaration from a normal user. New mode induction methods that allow *indeterminate* existence of the literals in clauses and also do not need novel user knowledge about domain should be introduced. In this study, the proposed system employs a new method that restricts the introduction of non-referenced (one-location) existential variables in the body of the clauses with requiring basic domain knowledge from user, based on the sound formalism in [26].

After determining the search strategy, search heuristics and hypothesis language, the last thing to decide is the representation form of the background

knowledge, logical form in a deductive database or tabular form in a relational database. Most ILP concept learning systems input background facts in Prolog language; this restricts the usage of ILP engines in real-world applications due to the time-consuming transformation phase of problem specification from tabular to logical format. The need for ILP engines that can be applied to tabular data is obvious.

As a result, in this study a concept learning ILP system is proposed, which employs relational association rule mining concepts and techniques. The system proposed is a hybrid (top-down/bottom-up) ILP system, which utilizes inverse resolution for generalization of concept instances in the presence of background knowledge and refines these general patterns into frequent and strong concept definitions with a modified APRIORI-based specialization operator.

Two versions of the proposed system are implemented, one of which is for learning definite clauses possibly with body predicates indirectly bounded to the head predicate and the other is for learning definite clauses with body predicates only directly bounded to the head predicate. In this study, Java programming language is utilized in the implementation. In the aim of improving the efficiency of database access and satisfying the common need of storing data in a relational database, the system employs IBM DB2 database as backend data storage.

1.3 Organization of the Thesis

Chapter 2 gives a general overview of Inductive Logic Programming, a detailed description of inverse resolution and association rule mining. Chapter 3 explains two version of the proposed method and compares the advantages and disadvantages of these two versions. Chapter 4 discusses the experimental results of the proposed method on three real-world problems. Finally, chapter 5 includes concluding remarks and possible improvements.

CHAPTER 2

INDUCTIVE LOGIC PROGRAMMING

2.1 General

When human brain reasons about events, it tries to prove and deduce the result with the help of assembled background knowledge about the event domain. So, how is background knowledge acquired and collected in human brain? Apart from the knowledge obtained from ancestors, human being collects some particular patterns recurring in different situations for similar future events. This ability of generalization from specific observations, called induction, influenced the development of Inductive Logic Programming (ILP) as a branch of Artificial Intelligence.

ILP basically studies learning concept definitions or regularities from specific instances in terms of prior known relations in clausal logic framework. Generally, ILP learner is presented a set of training examples and background knowledge in form of logic clauses, and induces concepts or frequent patterns as logical expressions. The term hypothesis is also used for induced concept/pattern description.

Inductive learning is in fact searching for complete and consistent concept descriptions in the space limited by description language of the ILP system [27]. The current state of art in ILP is achieving to find qualified logical hypothesis

efficiently, i.e. in minimal learning time. Current learning systems employ constraints on the search space via language, search strategy or user feedback in the sake of efficiency. These constraints are called bias [28].

Language bias is the limitations on the syntactic structure of the possible clauses in the hypothesis space. For instance, an ILP system may require the hypotheses to be definite clauses with at most n literals, etc. If more strict limitations are put on the description language, the search space will be smaller that results in an efficient learner. However, the restrictions may cause the learner to overlook some hypotheses of good quality. Therefore, an ILP system should balance the trade-off between the quality of hypotheses induced and the efficiency of the system.

After the borders of the search space are determined by language bias, the *search bias* should restrict which parts of the search space traversed according to a sound heuristics. The naive approach is to traverse the permitted clauses completely, one by one [29]. The efficiency considerations can not tolerate this exhaustive search; some filtering methods to prune the space should be utilized.

In some interactive systems, an oracle determines the soundness criteria of induced rules in the learning phase explicitly or implicitly [2]. These semantic rules imposed are called *declarative bias*. For example, the user determines the relations between the predicates in the background knowledge or guides the learner via deciding on the validity of the new predicates invented through search steps.

The biases, searching algorithm and quality criteria employed in evaluating induced hypotheses characterize the expressiveness and performance of an ILP system.

ILP systems can be classified into two general categories according to the learning task: predictive learning systems and descriptive learning systems. In predictive ILP systems, there is a specific target concept to be learned in the light of past experiences; however, there is no specific goal in descriptive learning and the task is to identify patterns in the data [30].

2.1.1 Predictive Inductive Learning

In predictive ILP, the task is learning concept/class descriptions, that correctly classify instances (and non-instances) of a specific concept, in terms of the background knowledge about the problem domain. Predictive learning can be applied to any classification or prediction problem, such as predicting carcinogenic activity of chemical compounds based on their chemical structures [31]. In this problem, the concept instance space is chemical compounds, the concept is whether a compound is carcinogenic or not and the task is finding correct classification rules that map positive instances to carcinogenic class and negative ones to non-carcinogenic class.

The problem setting of the predictive ILP learning task introduced by Muggleton in [32] can be stated as follows:

Given:

- Target class/concept C,
- A set E of positive and negative example of the class/concept C,
- A finite set of background facts/clauses B,
- Concept description language L (language bias).

Find:

- A finite set of clauses H, expressed in concept description language L, such that H together with the background knowledge B entail all positive instances E+ and none of the negative instances E-. In other words, H is complete and consistent with respect to B and E, respectively.

In this problem setting, completeness and consistency are the quality criteria for selecting the induced hypotheses; however the definitions of these terms require the hypotheses %100 fit the given instances, which is too strict for hypothesis to have predictive power. There may be errors in the background knowledge and training concept instances; or training examples can be sparse

to reflect the general regularities hidden in the concept [2]. Since success of a predictive learning system lies in the ability to generalize for unseen concept instances correctly, predictive ILP systems should employ more relaxed quality criterion that allow some training examples remain misclassified.

A predictive ILP system learns the target concept via searching hypothesis space in one of two directions: top-down and bottom-up. Bottom-up approach starts with the most specific clause containing a given positive example and generalizes the hypothesis until the concept description with the background knowledge implies all positive instances. On the other hand, a top-down ILP system begins with the most general hypothesis which covers all instances and non-instances of the concept and diminishes the borders of the hypothesis such that the final hypothesis covers no negative instance of the concept. MIS, FOIL and LINUS are among the top-down predictive ILP systems and CIGOL, GOLEM and PROGOL are among the bottom-up ones. Besides, a top-down (bottom-up) system may employ a generalization (specialization) operator in order to adapt the hypothesis according to given concept instances [33].

2.1.2 Descriptive Inductive Learning

Descriptive data mining differs from the predictive data mining such that the search is not directed by a target concept. A descriptive ILP system does not know which class or concept it is looking for in underlying database; instead it searches for interesting frequent patterns with no single target attribute, i.e. the consequent of the rules can be any attribute or relation in the data [34]. In other words, the data mining system explores relationships between the tendency of domain subjects in doing an action/having a property (buying a specific product/having cancer genetic effect) and domain-related features of the subjects (being female/having a specific molecular structure).

In descriptive data mining, the main objective is to find useful/interesting and understandable patterns. Therefore, the pattern representation language and the interestingness criterion play the main role in the success of a descriptive

data mining system.

There are two popular techniques in descriptive data mining, which are data summarization and clustering [30]. In data summarization field, WARMR and CLAUDIEN are the most popular descriptive ILP systems. Additionally, RDBC and FORC are well-known ILP systems based on relational distance-based methods to address the task of clustering.

2.2 Inverse Resolution

Bottom-up predictive ILP systems employ various generalization operators that start the search of the hypothesis space from the most specific clause, allowed by the hypothesis language, and generalize the clause until it can not be further generalized without covering negative concept instances [2].

A generalization operator maps a clause to a set of clauses which are the generalizations of the input clause based on the definition of generality relation/ordering utilized. Most ILP systems use the θ-subsumption based generality operators in the bottom-up search of clauses due to their simplicity and tractability. θ-subsumption relation between first-order clauses can be defined with a brief logic programming terminology abridged from [35] as follows:

term: A variable, constant or function symbol followed by n-tuple of terms

atom: Predicate symbol applied to terms

literal: Positive or negative atom

clause: A finite set of literals

existential/local variable: Variable that only occurs in the body of the clause[36]

global variable: Variable that occurs in the head of the clause[36]

definite clause: Clause containing exactly one positive literal

clausal theory: A set of clauses

well-formed formula: Literal, clause or clausal theory

substitution: $\theta = X_1/t_1, ..., X_k/t_k$ is a function from variables to terms. The application of a substitution to a well-formed formula results in replacing all occurrences of each variable X_i in the formula with the corresponding term t_i.

most general unifier (MGU): A substitution θ is a unifier for literals t_1 and t_2 such that $t_1\theta = t_2\theta$. The substitution θ is a most general unifier of literals t_1 and t_2 iff there is no unifier θ' for which the unified term $t_1\theta'$ is more general than $t_1\theta$. Two literals are unifiable if they have a MGU.

θ-subsumption: Clause C is at least as general as clause C$'$ if and only if C θ-subsumes C$'$, expressed as C $\leq$ C$'$. A clause C θ-subsumes a clause C$'$ if and only if there exists a substitution θ such that C$\theta \subseteq$ C$'$ [37]. Two clauses C and C$'$ are equivalent under θ-subsumption, if and only if C $\leq$ C$'$ and C$'$ $\leq$ C. Two clauses are variants, if they are equivalent up to variable renaming [38].

In other words, clause C θ-subsumes C$'$ if there is a substitution θ that can be applied to C such that every literal in Cθ occurs in C$'$. If $\theta = \emptyset$ and C $\leq$ C$'$, then C is a subset of C$'$; otherwise if $\theta \neq \emptyset$ and C $\leq$ C$'$, then C is a subset of C$'\theta^{-1}$ ($C\theta \subseteq C' \equiv C \subseteq C'\theta^{-1}$). Therefore, generalization operators under θ-subsumption perform two syntactic generalization operations: obtain C by applying inverse substitution to the clause C$'$ and/or removing one or more literals from the clause C$'$.

There are two basic subsumption based generalization operators: relative least general generalization developed by Plotkin [37] used in GOLEM and inverse resolution introduced by Muggleton and Buntine [21] used in CIGOL.

Inverse resolution is built on the fact that induction is the reverse operation of deduction. The resolution rule of deductive inference allows to derive a resolvent clause C entailed from two given parent clauses C_1 and C_2, employing SLD-resolution procedure [35]. The resolution rule has led inverse resolution

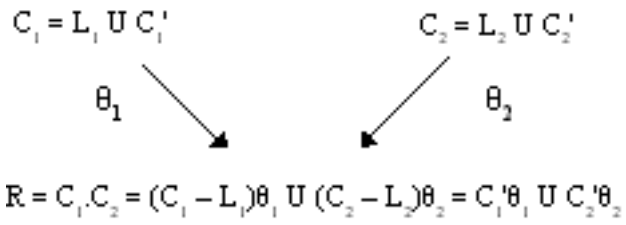

Figure 2.1: A single resolution step

to be introduced as a tool for induction that inverts the resolution process by generalizing C_1 from C and C_2 [21].

Resolution is used for inferring consequences of a clausal theory. Let's assume that the theory consists of two clauses C_1 and C_2 and the clauses contain the literals L_1 and L_2, such that θ is the MGU of $\neg L_1$ and L_2, respectively. The clauses C_1 and C_2 are resolved on the literals L_1 and L_2 when θ-substitution is applied to the union of the clauses. The resolvent R is derived as resolved product of C_1 and C_2, i.e. $R = C_1.C_2 = (C_1 - L_1)\theta \cup (C_2 - L_2)\theta$. Therefore, the theory θ-subsumes and entails the resolvent R.

The substitution θ can be uniquely factored into two exclusive substitution θ_1 and θ_2 such that $\theta = \theta_1\theta_2$, $vars(\theta_1) \subseteq vars(C_1)$, $vars(\theta_2) \subseteq vars(C_2)$ and $L_1\theta_1 = L_2\theta_2$. So, the resolvent becomes

$$R = (C_1 - L_1)\theta_1 \cup (C_2 - L_2)\theta_2 \quad (1)$$

In other words, resolution is able to derive the base clause of a '$\vee$' resolution tree, given two clauses on the arms as shown in Figure 2.1.

If there are more than two clauses in the theory, then the consequence will be derived by repeatedly applying resolution rule, resulting in a binary derivation tree.

Suppose that the background knowledge consists of the unit ground clauses male(joe), parent(joe, john) and the hypothesis is father(X, Y) :- parent(X, Y), male(X). The theory is the union of the background knowledge and the hypothesis. The problem is whether father(joe, john) can be derived from the stated theory. It is possible to find a derivation tree with the query fact as the

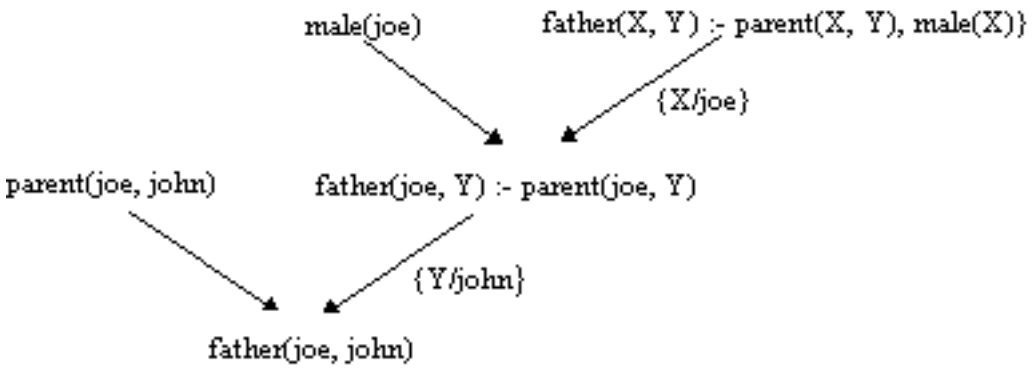

Figure 2.2: A first-order derivation tree

base clause:

- First, resolve the hypothesis and the first background fact. These two clauses resolve on the literals L_1 = male(joe) and L_2 = ¬male(X) such that $\theta_1 = \emptyset$ and θ_2 = X/joe. The resolvent C of C_1 = male(joe) and C_2 = father(X, Y) :- parent(X, Y), male(X) is $(C_1 - L_1)\theta_1 \cup (C_2 - L_2)\theta_2$ = father(joe, Y) :- parent(joe, Y).

- Secondly, resolve the consequence of the first derivation C with the second background fact. These two clauses resolve on the literals L'_1 = parent(joe, john) and L'_2 = parent(joe, Y) such that $\theta_1 = \emptyset$ and $\theta_2 = Y/john$. The resolvent of C'_1 = parent(joe, john) and C'_2 = father(joe, Y) :- parent(joe, Y) is $(C'_1 - L'_1)\theta_1 \cup (C'_2 - L'_2)\theta_2$ = father(joe, john). The binary resolution tree is shown in Figure 2.2.

Muggleton and Buntine employed three types of generalization operators based on inverse resolution in their CIGOL system: $\vee$-*operator* (*absorption operator*), W-operator and the truncation operator. The proposed system utilizes the $\vee$-operator in generalizing concept instances in the presence of background knowledge. Given C_1 and C, the $\vee$-operator finds C_2 such that C is an instance of the most general resolvent R of C_1 and C_2. As $(R \leq C)$, $\vee$-operator generalizes $\{C_1, C\}$ to $\{C_1, C_2\}$ [21].

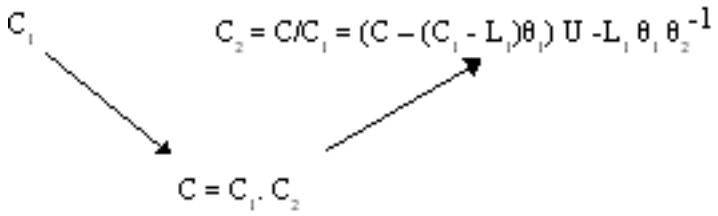

Figure 2.3: A simple ∨-operator step

In contrast to the resolution, the ∨-operator derives one of the clauses on one arm of the ∨ tree C_2, given the clause on the other arm C_1 and the base clause C as shown in Figure 2.3. From the notation of resolution $C = C_1.C_2$, it can be derived that $C_2 = C/C_1$ and C_2 is named as the resolved quotient of C and C_1 [21]. For the propositional case, the resolved quotient of two clauses is unique since there is no unification in propositional resolution that leads to indeterminacy, i.e. $\neg L_1 = L_2$. However, for the first-order case, it is not unique and can be derived as a result of the algebraic manipulation of the equation (1) as follows:

$$C_2 = (C \cup \neg(C_1 - L_1)\theta_1)\theta_2^{-1} \cup L_2 \quad (2)$$

Since $\theta_1\theta_2$ is the MGU of $\neg L_1$ and L_2, $\neg L_1\theta_1 = L_2\theta_2$ and thus:

$$L_2 = \neg L_1\theta_1\theta_2^{-1} \quad (3)$$

Substituting (3) into (2) as in Figure 2.3:

$$C_2 = (C \cup \neg(C_1 - L_1)\theta_1)\theta_2^{-1} \cup \neg L_1\theta_1\theta_2^{-1} = (C \cup \neg C_1\theta_1)\theta_2^{-1} \quad (4)$$

As C and C_1 are given as input, there are three unknown parameters, namely L_1, θ_1 and θ_2^{-1}, that lead to indeterminacy in equation (4). If the background knowledge C_1 is represented by ground unit clauses, i.e. $C_1 = L_1$ and $\theta_1 = \emptyset$ then equation (4) becomes:

$C_2 = (C \cup \neg L_1)\ \theta_2^{-1} = (C \cup \neg C_1)\ \theta_2^{-1}$ (5)

Therefore, the indeterminacy is reduced to the choice of the inverse substitution θ_2^{-1}. As the following definition of inverse substitution implies, selection of the inverse substitution θ_2^{-1} means the selection of the terms in $(C \cup \neg C_1)$

that should be mapped to distinct variables. The ILP learners employing inverse resolution as a generalization operator should apply a heuristic during the search of the inverse substitution space.

Inverse Substitution: Given a literal t and a substitution θ, there exists a unique inverse substitution θ^{-1} such that $t\theta\theta^{-1} = t$. Whereas the substitution is a function from variables to terms, the inverse substitution θ^{-1} is a function from terms to variables. If the substitution is $\theta = \{v_1/t_1, ..., v_n/t_n\}$, then the corresponding inverse substitution can be denoted by $\theta^{-1} = \{(t_1, \{p_{1,1}, ..., p_{1,m_1}\})/v_1, ..., (t_n, \{p_{n,1}, ..., p_{n,m_n}\})/v_n\}$ in which $p_{i,m}$ designates the places at which literal argument positions the term t_i is replaced by the variable v_i. The definition of places can be extended to 2-tuple form as [literal order, argument order] for representation of places within clauses.

In short, the objective of inverse resolution is to construct a derivation of a positive instance e1 by introducing new hypotheses which, together with old theory or background knowledge, entail e1. Applying $\vee$-operator to two clauses inverts a single resolution step, however applying the operator repeatedly results in inverting a whole derivation tree.

Suppose that an ILP system has the unit ground clauses male(joe), parent(joe, john) as the background knowledge. It is presented with a positive example father(joe, john) and asked to find a hypothesis such that the final theory consisting of the hypothesis entails father(joe, john). Let's try to find an inverse derivation tree with the positive example as the base clause:

- First, inverse resolve the positive instance with the first background fact. Since background fact is a ground unit clause, equation 5 applies. Therefore, $C_2 = (C \cup \neg C_1)\theta_2^{-1} = (\text{father(joe, john)} \cup \neg\text{male(joe)})\theta_2^{-1}$. One of the possible inverse substitutions is $\{joe/X\}$. If we apply this inverse substitution, the hypothesis $father(X, john)$:- $male(X)$ is generated.

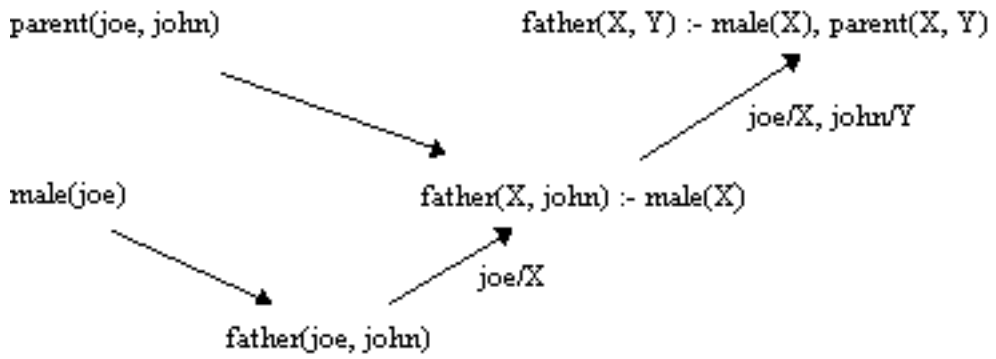

Figure 2.4: An inverse linear derivation tree

- Secondly, generalize the current hypothesis with the second background fact. Since background fact is also a ground unit clause, equation 5 applies. Therefore, $C'_2 = (C' \cup \neg C'_1)\theta_2^{-1} = (\text{father(X, john)} \cup \neg\text{male(X)} \cup \neg\text{parent(joe, john)})\theta_2^{-1}$. The inverse substitution can be $\{joe/X, john/Y\}$. So the hypothesis induced is $father(X,Y)$:- $male(X), parent(X,Y)$. The inverse linear derivation tree is illustrated in Figure 2.4.

In a real-world data mining problem, there are many positive instances of a concept and background facts related to that positive example, so that an ILP learner should decide on which background facts or positive instances to use.

In order to mine complete and consistent rules, a batch ILP system should select instances that correctly describe and sample the entire positive instance space. In other words, each chosen sample should reflect the general relation between the concept and background facts as the cluster of the instance space it belongs.

For each instance, a specific combination of the background facts result into an inverse derivation tree. Lets assume that the number of related background facts about the current positive instance is n. Then an ILP system, employing inverse resolution and performing exhaustive search, should traverse 2^n inverse resolution trees in order to find the most qualified rule(s). This search results in an inefficient learning system, therefore the ILP system should prune inconsistent or incomplete combinations of the background facts in a sound way.

Firstly, ILP system should not evaluate 2^n inverse resolution trees one by one separately, instead, it should apply the knowledge gained from the prior evaluations to the later one in order to minimize the search space. One possible method is to store the unqualified rules in order to bypass the repeated generation of these rules.

According to the definition of θ-subsumption, a subset of a clause subsumes and is as general as the original clause itself. As a result, if a sub-clause of a clause is not complete (does not entail a minimum amount of positive data), the clause, a specialization of its subset is not complete, either. This rule can be used as a pruning mechanism in such a way that, first one-element combinations of background facts are evaluated and incomplete ones are eliminated from scratch, then two-element combinations, generated from one-element ones, are evaluated and level-by-level evaluation continues until the stopping criterion is met. This method is in fact the popular APRIORI algorithm [39] utilized widely in association rule mining.

2.3 Association Rule Mining

Many organizations have lots of data about their customers, personnel, products, etc. The data mining systems aim at extracting knowledge about the stored data in favor of data owner for improving business gain. Lets consider a supermarket with a wide range of product. The product manager of the market should decide on the locations of the products on the shelves, which products not to put on the sale or which products to be sold at a discount in order to maximize the attractiveness of the products. The best way to make good decisions relies on the uncovered relations between products (items) in the customer transaction data collected so far. For instance, the manager can use the information about the products that people buy together to position items, to decide the products to be sold at a discount or to decide whether a product should be removed from the shelves or not. The general objective of association rule mining is to find frequent associations built-in the subsets of the data and enhance the functionality of

databases in a way that decision makers can query such associations. The most popular application area of association rule extracting systems is market basket type transactional databases. Many algorithms have been developed in order to find interesting Boolean association rules between sets of basket items [40, 41, 42, 43].

2.3.1 Boolean Association Rule Mining

Boolean association rules are in fact propositional classification rules with no single target attribute, in the form of X $\Rightarrow$ Y where X and Y are a set of conditions with no restriction on the consequent Y [30]. The relations between the attributes of a single table are represented by boolean association rules. For example, assume that there is a table of personnel working in a company with attributes age, sex, # of children, income, title, graduate degree etc. An example association rule is "graduate degree $\geq$ master $\Rightarrow$ title = specialist" about the Personnel relation. This rule shows that personnel who have a master degree or higher tend to be in a position Specialist or higher in the company.

The most popular application area of the boolean association rule mining is the market basket problem. In the market basket problem, the database consists of a transaction table with columns, each of which represents a product type on sale, and rows representing baskets that store items purchased on a transaction. The goal of market-basket mining is to find strong association rules between frequent item sets [44]. A sample association rule is "Coke $\Rightarrow$ Chips", which can be interpreted as "Coke causes chips to be bought".

A frequent item set is defined as an item set with support value greater than a support threshold, s. *Support* of an item set is the frequency of the item set, defined in terms of the fraction of the baskets including item set. For instance, if most of the baskets in the transaction base (rows in the table) consist of Coke and Chips, then Coke, Chips is a frequent item set.

Support of an association rule X $\Rightarrow$ Y is the support of the item set X $\cup$ Y [45]. If the support of a rule is smaller than the threshold, then the rule can

only explain the tendency for very small fraction of the transactions and it is unnecessary to take it into consideration for future business plans.

A strong association rule is a rule that has confidence greater than a confidence threshold, c. Confidence of an association rule $X \Rightarrow Y$ is evaluated by the probability of the baskets, having the item set X, also have items Y. In other words, the confidence of an association rule is the ratio of the number of baskets that contain the item set $X \cup Y$ to the number of baskets including the item set X [45]. If most of the baskets including Coke also include Chips, then Coke $\Rightarrow$ Chips is a strong association rule.

Support shows the generality of the rule and the confidence designates the validity of the rule. If Coke, Chips is a frequent item set and Coke $\Rightarrow$ Chips is a strong association rule, then Coke $\Rightarrow$ Chips is valid for most of the baskets in database.

Market basket problem can be formally defined as follows : Given a set of items $I = \{I_1, I_2, I_3 \cdots I_n\}$ and a table of transactions $T = \{t_1, t_2, t_3 \cdots t_m\}$ where each transaction is represented as $t_i = \{I_{i1}, I_{i2} \cdots I_{ik}\}$ and $I_{ij} \in I$, the association rule problem is to find all association rules $X \Rightarrow Y$ with support and confidence above thresholds [45].

The common approach in market-basket area is first finding frequent item sets and then deriving strong association rules from these frequent item sets. In this approach, the main critical point is to extract frequent item sets from transaction database efficiently. The naive method is to count all combinations of items in the portfolio that appear in the transactions. If the number of item types in the shelves is n, then each of $2^n - 1$ item sets should be counted against the database, which is exponential and costly. Therefore, most association rule mining algorithms develop smart methods in order to reduce the number of item sets to be counted [45]. The most popular and well known association rule mining algorithm, as introduced in [41], is APRIORI.

Table 2.1: An example Prolog database

person(seda).	sibling(seda, eda).	doctor(eda).
person(guzen).	sibling(seda, korkut).	doctor(korkut).
person(serkan).	sibling(guzen, aysu).	manager(aysu).
	sibling(serkan, serap)	teacher(serap)

2.3.2 Relational Association Rule Mining

Association rule mining aims at discovering hidden structures, also called patterns, in data. In boolean association rule discovery, there is one object type and one database table describing different features of this object type. The patterns mined are feature sets that are common for number of objects exceeding a frequency threshold. For instance, in the market-basket problem, the objects are baskets, each item is one feature of the basket and the patterns are the frequent item sets common in baskets.

In relational association rule mining, there are more than one object type and the patterns are not only feature sets but also they consist of relations between objects. Relational association rule mining can be described as discovering recurrent relational patterns in a relational database.

2.3.2.1 Relational Patterns

Feature/Item sets are not capable of representing the features of different objects and relations among them. The propositional representation of item sets should be upgraded to predicate sets in first-order logic framework. In first-order logic, each relation is represented by a predicate and the objects about which the relation is made are represented by variables in predicates. The predicate sets are in fact first-order queries; and the main task in relational association rule mining is to discover the interesting queries that best match the database.

For example, we have a relational database including five relations represented by Prolog facts in Table 2.1.

An example relational pattern for the above database is “people whose sibling

is a doctor". We can translate this query as a predicate set as follows: person(X), sibling(X, Y), doctor(Y)

It can be derived that the queries are in fact item sets in which items are related by variables. In other words, they are "relational item sets" [46].

Since relational item sets can include more than one object/variable, it is not clear which object or object pairs/tuples are counted in order to evaluate the frequency of patterns. The key predicate of counting should be determined in the formulation of frequent relation pattern discovery problem. For Q_1, there are three alternative object sets to be counted: Only people (X), only doctors (Y) or sibling-sibling pairs (X, Y). For each alternative, the meaning of the pattern changes:

- If we count X, then it means we try to discover "people whose sibling is a doctor". The key predicate is "person".
- If we count Y, the pattern searched is "doctors who has a sibling". The key predicate is "doctor".
- If we count X-Y, it means the pattern is "two siblings one of whom is a doctor" and the key predicate is "sibling".

As a result, a relational pattern can be represented by a first-order query and the key predicate with its key fields.

The support of a first-order query Q is defined as the number of instantiations of key fields for which query Q succeeds against the database. Since the support of a pattern is defined as relative, the frequency of the query is divided by the total number of instantiations of key fields in the key predicate.

Let's reconsider the query Q_1. The answer set X = seda will be retrieved if we run the query Q_1 against the above database, so the support of the query Q_1 is 1. Since the key predicate is "person", the total number of instantiations for the variable X is 3 as the answer set for query person(X) is X = seda, X = guzen, X = serkan. Thereafter, the support of the pattern Q_1 is 1/3 = 33%. That means that 33% of people in the database have a sibling who is a doctor.

2.3.2.2 First-order Association Rules

Relational association rules are the causality relations between different structural features of key objects. As in relational patterns, the key object(s) that the rule is about mainly determines the meaning of the association rule.

In boolean association rule mining, the meaning of a Boolean association rule is unique since there is only one object in database. For instance, in market basket problem the object is basket and an association rule, formulated as X $\Rightarrow$ Y, means that "if a basket includes the item set X, then it is most probable that it also includes the item set Y". The key object of the association rules is the same, by default, the basket.

However, in relational association rule mining, it is possible to generate association rules that contain objects other than the key objects and the head of the rule does not include key object or objects. For instance, we can conclude the following association rule from the predicate set person(X), sibling(X, Y), doctor(Y) where "person" is the key predicate:

$person(X), sibling(X, Y) \Rightarrow doctor(Y)$

Since the key predicate is "person", the object that will be counted is X. The rule includes the sibling Y as an object other than the key object and the key object X is not in the head of the rule. As a Prolog clause, it can be read that "If a person has a sibling, then this sibling is a doctor"; however, the rule, as an association rule, relates a person's feature of having a sibling with the feature of the same person having a sibling who is a doctor. In the first interpretation the key object is the sibling; in the latter, it is the person [1].

The difference in meaning of relational rules in predictive and descriptive framework results in difference in the notation of relational rules. While relational rules are shown as $X \Rightarrow Y$ in Prolog form, the association rules, called "query extensions", are shown as $X \sim> Y$ [46].

The meaning of relational association rules is the same as the original meaning of the Prolog definite clauses when the key predicate is the predicate in the head of the rule and the key variables of the association rules occur in the head.

For instance, in the above rule if the key variable is Y, then the meaning of the association rule would be "If a person has a sibling, then it is most probable that this person is a doctor." It is the same as "If a person has a sibling, then this sibling is a doctor". Therefore, an association rule finder can be used as a predictive data mining system with support concept if the system guarantees that the head predicate of the rule is the key predicate and a subset of the variables in the head consists of the key variables.

The support of a relational association rule is the support of the predicate set in the rule. The confidence of a relational association rule $X \sim> Y$ is defined as the support of the pattern $X \cup Y$ divided by the support of the body X. The confidence of the association rule $person(X), sibling(X, Y) \sim> doctor(Y)$, with the key object X, can be computed according to the database in Table 2.1 as follows: The frequency of $person(X), sibling(X, Y), doctor(Y)$ is 1/3 as evaluated. The frequency of $person(X), sibling(X, Y)$ is 1/1 since all people have a sibling. Therefore, the confidence of the rule is 1/3 / 1/1 = 1/3.

2.4 APRIORI

APRIORI utilizes an important property of frequent item sets in order to prune candidate item set space:

All subsets of a frequent item set must be large.

The contra positive of this property says that if an item set is not frequent then any superset of this set is also not frequent. It can be concluded that the item set space should be traversed from small size item sets to large ones in order to discard any superset of infrequent item sets from scratch. In order to apply this reasoning, APRIORI reorganizes the item set space as a lattice based on the subset relation, as shown in Figure 2.5.

The item set lattice in Figure 2.5 is composed of possible large item sets for items I_1, I_2, I_3. The directed lines in the lattice represent the subset relationships, and the frequent item set property says that any set in a path below an item set is infrequent if the original item set is infrequent. For instance, if

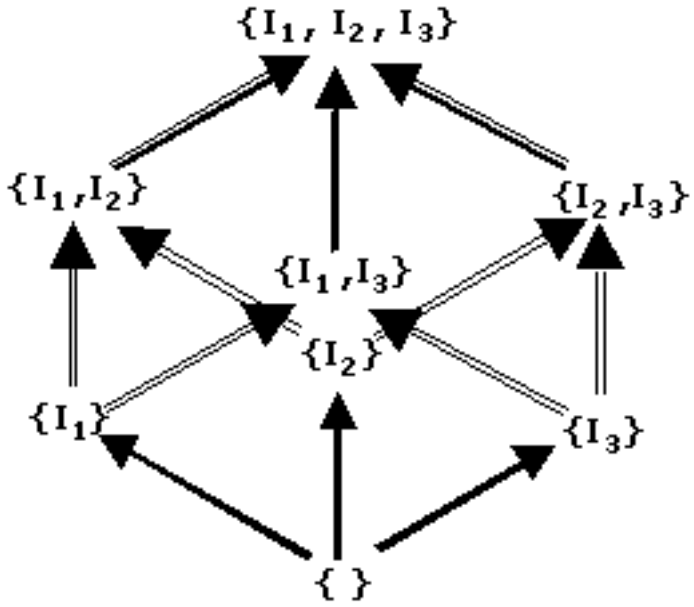

Figure 2.5: The APRIORI lattice with three items

the item I_1 is not found frequently in transaction baskets, then the item sets $\{I_1, I_2\}, \{I_1, I_3\}$ and $\{I_1, I_2, I_3\}$ are not frequent, either.

In APRIORI, an item set is called a candidate if all its subsets are frequent item sets. An item set is large/frequent if it is candidate and the number of occurrences of this item set in transactions is greater than the support threshold value.

APRIORI algorithm proceeds levelwise in the lattice as follows:

Step 1. All item sets of size 1 (items itself, I_1, I_2, I_3) are used as candidate item sets, C_1, in the first step. Find large item sets from C_1 that appear at least fraction s (support threshold) of baskets. This set of large item sets is expressed as L_1.

Step 2. Generate n+1-element candidate item sets C_{n+1} from n-element large item sets L_n by combining n-element large item sets that have n-1 items in common.

Step 3. Scan the database to count n+1-element candidate item sets in transactions and decide if they are large. The resultant set of n+1-element large

item sets is L_{n+1}. Go to step 2 if L_{n+1} is not empty set, otherwise go to step 4.

Step 4. Output $L_1 \cup L_2 \cup \cdots \cup L_n$.

As explained in the algorithm, APRIORI makes one database scan per level. This results in n+1 database scans, which is costly if the item set lattice is too deep. In order to prevent this weakness of the APRIORI algorithm, most data miners limit the maximum cardinality of a possible frequent item set.

2.5 WARMR

In [47], a relational association rule miner that discovers frequent Datalog queries, WARMR, is presented. WARMR takes a Datalog relational database and a support threshold as input and outputs Datalog queries that are frequent in the input database. Since first-order predicate language allows the use of variables and multiple relations in patterns, the patterns are more expressive than the propositional ones; besides, the size of the pattern space is huge.

A relational association rule miner should determine a formalism to syntactically constrain the query language to a set of meaningful queries. For instance, the formalism/declarative bias should exclude queries that bind incompatible argument types, like unifying a person and a product type variables in "sibling(X, Y), buys(john, Y)" (Y is a person in "sibling" predicate and is a product in "buys" predicate).

In the language formalism of WARMR, WRMODE, a set of all possible ground and non-ground atoms is explicitly presented to the system. Each variable argument of each atom in the set is marked by means of three mode-labels +, - and $\mp$; where + means that the variable is strictly input/bound, i.e. has to appear earlier in the query; - means that the variable is strictly output/unbound, i.e. must not appear earlier; $\mp$ means that the variable can be both input and/or output.

Input-output modes of the variables in the formalism constrain the refinement of queries in a way that the modes determine which atoms can be added to a query [1]. The key predicate of frequent patterns is specified in the formalism, too. Additionally, the types of the variables can be declared as in PROGOL [23]. An example declarative language bias specification in WRMODE notation for the database in Table 2.1 is illustrated below.

Key = person(-) Atoms = sibling(+, -), doctor(+), manager(+), teacher(+)

As an analogy to the APRIORI trick, each meaningful sub-query, that the declarative language bias allows, of a frequent query should be a frequent query. The WARMR algorithm, as shown in Table 2.2 [46], thus employs a level-wise search such that specific candidate queries Q2 are generated from simpler, general frequent queries Q1 where Q1 -subsumes Q2.

Table 2.2: The pseudo code of the WARMR Algorithm

Inputs: database r; WRMODE language L; support threshold *minfreq*
Outputs: all queries Q $\in$ L with frequency $\geq$ *minfreq*

1.Initialize level d := 1
2.Initialize the set of candidate queries Q1 := ?- key
3.Initialize the set of infrequent queries I := {}
4.Initialize the set of frequent queries F := {}
5.While Q_d not empty
  a.Find frequency of all queries Q $\in$ Q_d
  b.Move the queries with frequency $\leq$ *minfreq* to I
  c.Update F := F $\cup$ Q_d
  d.Compute new candidates Q_{d+1} from Q_d,
  F and I using WARMR-GEN
  e.Increment d by 1.
6.return F

Function: WARMR-GEN(L; I; F; Q_d);

1.Initialize Q_{d+1} =
2.For each $Q_j \in Q_d$, and for each refinement $Q_j \in$ L of Q_j:
  a.Check whether Q_j is θ-subsumed by some query $\in$ I, and
  b.Check whether Q_j is equivalent to some query in $Q_{d+1} \cup$ F
  c.If both are not true, add Q_j to Q_{d+1}.
3.return Q_{d+1}.

WARMR starts with the query ?− key at level 1 and generates query candidates C_{l+1} at level l+1 by refining frequent queries F_l obtained at level l. The frequency of candidates C_{l+1} are evaluated against the database; the queries that have frequencies above the threshold value are moved to F_{l+1}. This candidate generation and evaluation loop continues until no more candidate query is produced.

The main difference of WARMR from APRIORI is the candidate generation step where queries are refined by adding one atom to the query at a time as allowed by the mode and type declarations, instead of combining frequent subqueries as in APRIORI. This is due to the fact that all generalizations of a frequent query may not be in the language of admissible patterns determined by declarative bias; and frequent queries that have sub-queries not in the declarative language will not be discovered. Therefore, the built-in pruning of search space in APRIORI should be done explicitly by WARMR. The relational algorithm explicitly keeps track of the infrequent queries and checks whether the candidate query is a specialization of an infrequent query during every candidate generation step.

Given the mode declarations and the database in Table 2.1, WARMR starts the search of frequent queries at level 1 with the key predicate person(X). The literals {sibling(X, Y), doctor(X), manager(X), teacher(X)} can be added to the query at level 1 yielding level 2 candidate queries {(person(X), sibling(X, Y)), (person(X), doctor(X)), (person(X), manager(X)), (person(X), teacher(X))}. Taking the first of level 2 candidate queries, the formalism allows {sibling(X, Z), sibling(Y, Z), doctor(X), doctor(Y), manager(X), manager(Y), teacher(X), teacher(Y)} to be added to obtain level 3 candidate queries. The refinement graph will continue to grow until no more candidates are remained.

After WARMR discovers all frequent queries and their frequencies, these patterns can be post-processed into relational association rules that exceed given confidence threshold value. Since relational association rules are couples of queries that one extends the other, it is sufficient to find frequent query couples

(Q_1, Q_2) such that Q_1 extends Q_2 and conclude query extensions $Q_2 \sim> Q_1 - Q_2$. As an example, if $Q_1 = person(X), sibling(X, Y), doctor(Y)$ with $support = 1/3$ and $Q_2 = person(X), sibling(X)$ with $support = 1/1$, then the resultant query extension is $person(X), sibling(X) \sim> doctor(Y)$ with $confidence = 1/3$.

The main advantage of the WARMR system is its flexibility offered to the user in determining the search space of possible patterns and adding background knowledge to the database. These settings are fully isolated from the implementation. However, the mode declaration in the formalism is too hard for a normal user to state which patterns he/she really wants to discover and it is not practical for large databases. It should not be overlooked that the user in descriptive data mining does not know what he/she wants to find and even does not have deep knowledge about relations in database. Therefore, the system should be capable of pruning the search space without mode declarations, perhaps, with only type declarations.

Explicit pruning step of WARMR requires storing a list of infrequent patterns and traversing this list for every candidate query. The size of the infrequent query set increases as the database structure becomes complicated and this pruning step increases time complexity of WARMR with respect to APRIORI.

CHAPTER 3

THE PROPOSED SYSTEM

Concept definition is in fact making associations between the concept and the preconditions to this concept. We propose in this study a concept learning ILP technique, which employs relational association rule mining techniques. The technique proposed utilizes inverse resolution for generalization of concept instances in the presence of background knowledge and refines these general patterns into frequent and strong concept definitions with a relational upgrade of the APRIORI refinement operator. In this chapter, two versions of this technique, which differ in syntactic bias, are discussed in detail.

3.1 Language Bias and Search Space

A concept is in fact a set of frequent patterns, embedded in the features of the concept instances and relations of objects belong to the concept with other objects. Since the predicate calculus is capable of representation of relations via predicates and relations between predicates via shared variables among predicate arguments, first-order logical framework is chosen as the concept definition language where concepts/relational patterns are represented by function-free definite clauses. A clause can be interpreted as a partial definition for a concept, where the head predicate identifies the defined concept and the predicates

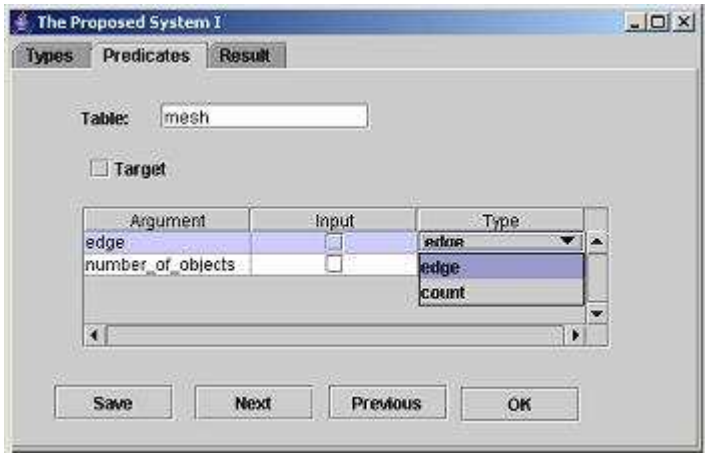

Figure 3.1: The graphical user interface of the Proposed System I

in the body of the clause represent the required features and relations for an object that belongs to this concept [48].

However, first-order logic as a pattern language allows unreasonable clauses in the search space, such as patterns where features having different data types are related. Therefore, the proposed system puts an additional constraint on the clause space in such a way that only predicate arguments, which have the same data type, can be unified with the same variable name. For example, lets assume input relational database is composed of "buy", "customer" and "product" facts with type declarations buy (person, item), customer (person), product (item, color). Then clause

$$buy(X, Y) : -customer(X), product(Y, red)$$

is in the pattern language, however the clause

$$buy(X, Y) : -customer(X), product(X, red)$$

is not in the search space since the first arguments of the predicates "customer" and "product" have incompatible types. The user supplies the type declarations of predicate arguments via the system's graphical user interface in Figure 3.1. The user also supplies the concept to be learned via the interface.

The system imposes a generality ordering on the concept clauses based on θ-subsumption, which guides the search in generating candidate concept clauses. The generality relation between clause examples of the same concept, used in this work, is a restricted version of θ-subsumption relation expressed as follows: [48]

A definite clause C θ-subsumes a definite clause C$'$, i.e. is at least as general as C$'$, if and only if $\exists\theta$ such that:

- $head(C) = head(C')$

- $body(C') \supseteq body(C)\theta$

In other words, the body of the clause C unifies with a subpart of body of the clause C$'$.

The system utilizes "support" besides the generality ordering in pruning the search space. In analogy to the frequent query property in WARMR, we adopt the fact that "Each meaningful generalization of a frequent clause, with the same head predicate, should also be a frequent clause". The important point in this trick is that the head of the generalizations should be the same as the original clause in order to compare their frequencies since the head predicate is the key predicate of the pattern and the key predicate designates the object(s) to be counted.

The contra-positive of the above property of definite clauses says that if a clause is infrequent, it is redundant to generate any specialization of it as candidate concept clauses since any specialization of an infrequent clause is also infrequent. The APRIORI lattice in Figure 2.5 can be used to illustrate the structure of the clausal search space, where each letter represents a clause instead of an item set and the lines in the lattice represent the generality relationship between clauses. Any clause in a path above a clause must be frequent if the original clause is frequent.

3.2 Proposed System I

The system proposed employs a coverage algorithm in constructing concept definition. It first selects a positive concept example, based on the order of concept instances in the database. The most general clauses, with two predicates, that entail the positive example are generated and then the concept rule space is searched with an APRIORI-like specialization operator. The specialization operator utilizes the frequency property of definite clauses in order to effectively prune the search space. Among the frequent and strong rules produced, the system selects the best clause using a criterion called *f-measure* [49], which is discussed later, and repeats the rule search for the remaining concept instances that are not in the coverage of the hypothesis clauses. The overall pseudo code of the system is given in Table 3.6.

The proposed system will be explained in three basic sections: generalization, refinement and evaluation.

3.2.1 First Phase: Generalization of Positive Examples

Throughout the chapter, we will use the *daughter* relation in Table 3.1 as a running example. Two versions of the proposed system will be explained on this learning task.

Table 3.1: The database of the daughter example with type declarations

Concept Instances	**Background Facts**	**Type Declarations**
daughter(mary, ann). daughter(eve, tom).	parent(ann, mary). parent(ann, tom). parent(tom, eve). female(ann). female(mary). female(eve).	daughter(person, person). parent(person, person). female(person).

After picking a positive example, the system searches facts related to the concept instance in the database, including the related facts that belong to the target concept in order for the system to induce recursive rules. Two facts are

related if they share the same constant in the predicate argument positions of same type.

For each related fact, the system derives concept descriptions (CD) that generalize the behavior of the concept instance (CI) in terms of the related fact (RF). The system utilizes the absorption operator, defined in Section 2.2, to form a V-tree for each concept instance-related fact couple (CI, RF) and derives all possible generalizations CD on one arm of the V-tree, given the concept instance CI as the base clause and the related fact RF on the other arm of the tree.

$$CD = (CI \cup \neg RF\theta_1)\theta_2^{-1}$$

Since the database fact RF is ground, θ_1 is empty substitution and the system searches for generalizations of the form $(CI \cup \neg RF)\theta_2^{-1}$, i.e. all possible values of inverse substitution θ_2^{-1}.

In short, concept instance and each related fact are generalized into two literal definite clauses, in such a way that the concept instance is an instance of the most general resolvent of the related fact and each generalization. For each target concept fact related to the original concept instance, two literal recursive generalizations are obtained.

In the daughter example, proposed system I selects the first concept instance "daughter(mary, ann)" and then finds the related fact set of the current concept instance {parent(ann, mary), parent(ann, tom), female(ann), female(mary)}. In the generalization phase, the system generalizes the concept instance "daughter(mary, ann)" in the presence of each related fact via absorption operator. Now it will be shown how the system generalizes the concept instance and the first related fact "parent(ann, mary)" into two predicate definite clauses. Applying absorption operator, the concept descriptions of the form $\{daughter(mary, ann) : -parent(ann, mary)\}\theta_2^{-1}$ are derived. The table 3.2 consists of the possible inverse substitutions and the resultant concept descriptions. In the table, the inverse substitutions are in the form of (term, the locations of the term in the clause)/variable where the locations of the term in

the clause are represented with 2-tuples such that first entry represents the order of the predicate in the clause and the second entry shows the argument order in the predicate. (Both predicate and argument order start with 0 instead of 1)

Table 3.2: Two predicate concept descriptions generated by the Proposed System I

Concept Description	**Value of θ_2^{-1}**
daughter(X3,ann):-parent(ann,X3)	{(mary,[0, 0][1, 1])/X3}
daughter(X1,ann):-parent(ann,X2)	{(mary,[0, 0])/X1,(mary,[1,1])/X2}
daughter(mary,X6):- parent(X6,mary)	{(ann,[0, 1][1, 0])/X6}
daughter(mary,X4):- parent(X5,mary)	{(ann,[0, 1])/X4,(ann,[1, 0])/X5}
daughter(X1,ann):-parent(X5,mary)	{(mary,[0, 0])/X1,(ann,[1, 0])/X5}
daughter(X1,X6):-parent(X6,mary)	{(mary,[0, 0])/X1,(ann,[0, 1][1,0])/X6}
daughter(X1,X4):-parent(X5,mary)	{(mary,[0, 0])/X1,(ann,[0, 1])/X4, (ann,[1,0])/X5}
daughter(mary,X4):-parent(ann,X2)	{(mary,[1, 1])/X2,(ann,[0,1])/X4}
daughter(mary,X6):-parent(X6,X2)	{(mary,[1, 1])/X2,(ann,[0, 1][1,0])/X6}
daughter(mary,X4):-parent(X5,X2)	{(mary,[1, 1])/X2,(ann,[0, 1])/X4, (ann,[1, 0])/X5}
daughter(X3,X4):-parent(ann,X3)	{(mary,[0, 0][1, 1])/X3,(ann,[0,1])/X4}
daughter(X3,ann):-parent(X5,X3)	{(mary,[0, 0][1, 1]/X3), (ann,[1, 0])/X5}
daughter(X3,X6):-parent(X6,X3)	{(mary,[0, 0][1, 1])/X3, (ann,[0, 1][1, 0])/X6}
daughter(X3,X4):-parent(X5,X3)	{(mary,[0, 0][1, 1])/X3,(ann,[0, 1])/X4, (ann,[1, 0])/X5}
daughter(X1,X4):-parent(ann,X2)	{(mary,[0, 0]/X1),(mary,[1, 1])/X2, (ann,[0, 1])/X4}
daughter(X1,ann):-parent(X5,X2)	{(mary,[0, 0])/X1,(mary,[1, 1])/X2, (ann,[1, 0])/X5}
daughter(X1,X6):-parent(X6,X2)	{(mary,[0, 0])/X1,(mary,[1, 1])/X2, (ann,[0, 1][1, 0])/X6}
daughter(X1,X4):-parent(X5,X2)	{(mary,[0, 0])/X1, (mary,[1, 1])/X2, (ann,[0, 1])/X4, (ann,[1, 0])/X5}

The generalization phase of the implementation is given in pseudo code form in Table 3.3.

Table 3.3: The pseudo code of the generalization operator in the Proposed System I

```
relatedBackgroundFacts = sampleTargetFact.getRelatedFacts;
for each fact in relatedBackgroundFacts
do
      1.Construct the most specific clause with two literals
      that entails sampleTargetFact (sampleTargetFact ← fact)
      2.Generate two predicate θ-subsumption generalizations of
      this most specific clause, i.e. find all possible inverse substitutions
      using inverse resolution V-operator.
end do
```

3.2.2 Second Phase: Refinement of Generalizations

After the generalization phase, the system populates first level of the APRIORI lattice with two predicate concept descriptions obtained in the first phase. In the second phase, the system refines the two predicate concept descriptions with an APRIORI-based specialization operator that searches the definite clause space in a top-down manner, from general to specific.

As in APRIORI, the search proceeds level-wise in the hypothesis space and it is mainly composed of two steps: frequent clause set selection F_l from candidate clauses C_l and candidate clause set generation C_{l+1} as refinements of the frequent clauses F_l in the previous level. We extend standard APRIORI search lattice to capture first-order logical clauses and customize the candidate generation and frequent pattern selection tasks for first-order logical clauses.

3.2.2.1 Frequent Clause Selection

After candidate clauses are generated, the system picks the frequent clauses in the candidate clause set based on the support value of clauses. If the support value of a clause is greater than the support threshold, then the clause is refined otherwise eliminated, as in APRIORI.

In the section 2.3.2, the support computation for relational association rules is illustrated against a deductive database. Since a relational database is used in

place of a deductive one in the implementation phase, the support of a definite clause should be expressed in relational query language terminology, SQL syntax.

The absolute *support* value of a definite clause C, the number of key objects having the structure represented by the relational pattern in C, can be obtained with the following SQL query [46]:

```
SELECT count(distinct *) FROM
        SELECT key fields in the head/concept predicate
        FROM relations in C
        WHERE conditions expressed in the clause C
```

The relative support value is computed by dividing the absolute support value by the total number of key objects:

```
SELECT count(distinct *) FROM
        SELECT key fields in the head/concept predicate
        FROM the head relation in C
```

For instance, the following formula computes the relative support value of the clause "daughter(X, Y):- parent(Y, X)" for the daughter example:

```
Relative Support = Count1 / Count2
where count1 is found as
SELECT count(distinct *) FROM
        SELECT daughter.arg1, daughter.arg2
        FROM daughter, parent
        WHERE daughter.arg1 = parent.arg2 AND
        daughter.arg2 = parent.arg1
and count2 is found as
SELECT count(distinct *) FROM
        SELECT daughter.arg1, daughter.arg2
        FROM daughter
```

According to the database in the daughter example, this query returns the support value of $2/2 = 1$. In Table 3.4, the relative support value of each generalization in Table 3.2 is illustrated.

Table 3.4: The relative support values of two literal concept descriptions generated in the first phase

Concept Description	**Relative Support Value**
daughter(X3,ann):-parent(ann,X3)	0.5
daughter(X1,ann):-parent(ann,X2)	0.5
daughter(mary,X6):-parent(X6,mary)	0.5
daughter(mary,X4):-parent(X5,mary)	0.5
daughter(X1,ann):-parent(X5,mary)	0.5
daughter(X1,X6):-parent(X6,mary)	0.5
daughter(X1,X4):-parent(X5,mary)	1.0
daughter(mary,X4):-parent(ann,X2)	0.5
daughter(mary,X6):-parent(X6,X2)	0.5
daughter(mary,X4):-parent(X5,X2)	0.5
daughter(X3,X4):-parent(ann,X3)	0.5
daughter(X3,ann):-parent(X5,X3)	0.5
daughter(X3,X6):-parent(X6,X3)	1.0
daughter(X3,X4):-parent(X5,X3)	1.0
daughter(X1,X4):-parent(ann,X2)	1.0
daughter(X1,ann):-parent(X5,X2)	0.5
daughter(X1,X6):-parent(X6,X2)	1.0
daughter(X1,X4):-parent(X5,X2)	1.0

3.2.2.2 Candidate Clause Generation

After the frequent clauses to be refined are selected, candidate clauses for the next level of the search space are generated. Candidate clause generation is composed of three important steps:

- The set of frequent clauses of the previous level, F_{i-1}, is joined with itself to generate the candidate clauses C_i via union operator.
- For each candidate clause, a further specialization step is employed that unifies the existential variables of the same type in the body of the clause.
- The candidate clauses that have a partially connected structure are eliminated via filtering.

In order to apply the union operator to two frequent definite clauses, there are three conditions for these clauses to fulfill. First, these clauses must have

the same head literal and bodies that have all but one literal in common in order to be combined. Since only clauses that have the same head literal are combined, the search space is partitioned into disjoint APRIORI sub-lattices according to the head literal. Secondly, the system does not combine clauses that are specializations of the same candidate clause produced in the second step of the candidate generation task in order to prevent repeated literals in the body of the clauses and rapid expansion of the search space. For instance, the clauses "samegeneration(X2,X1):-samegeneration(X3,X5), parent(X5,X2)" and "samegeneration(X2,X1):-samegeneration(X5,X5), parent(X5,X2)" are specialized from the clause "samegeneration(X2,X1):-samegeneration(X3,X5), parent(X9,X2)" by unifying existential variables in the second step of the candidate generation and are not suitable to be combined. Finally, recursive clauses are forbidden to be combined in order to prevent rapid expansion of the search space, i.e. the system does not allow recursive clauses except linearly recursive ones.

If the frequent clauses are suitable to be combined, the union of the clauses is computed with the relational extension of APRIORI concatenation operator:

$$C_1 \cup C_2 = \{C_1 \cup \ l_{21} \mid C_{12} = C_1 \cap C_2\theta \ \wedge \ C_2\theta - C_{12} = l_{21}\}$$

As shown above, union of two clauses is in fact appending the literal in C_2 but not in C_1 to the body of the clause C_1. Before appending, the system applies a substitution to the literal in order to rename variables in the literal according to the variables in the first clause C_1. Since there may be more than one intersection of two frequent clauses, it is possible to produce multiple unions of two clauses.

If the concept descriptions are refined with only union operator, the search space will consist of clauses that have body relations/predicates directly bound to the head predicate through head variables. The structure of such clauses can be figured out as in Figure 3.2.

However, the system should also find clauses that have body predicates indirectly (not directly) bound to the head predicate. These clauses are needed

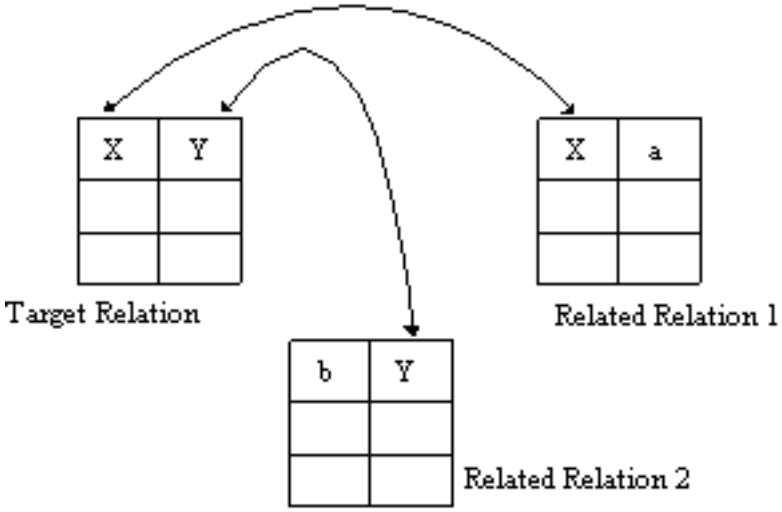

Figure 3.2: A clause with body predicates directly bound to the head predicate

to represent structured properties of the key objects. Body predicates bound by existential variables represent the structured properties of key objects in the logical framework, as illustrated with dotted lines in Figure 3.3.

In order to capture clauses that have relations indirectly bound to the head predicate, the system applies a further specialization operator to each union clause in the candidate generation task. The specialization operator unifies the existential variables of the same type in the body of the clause. We define the specialization operator formally as follows:

1. First the operator finds all existential variables in the union clause, that occur only once.

$$E(c) = \{v \in body(c) \mid occurs(v, c) = 1\}$$

2. Then it finds all pairs of existential variables that have the same type.

$$P(c) = \{(v1, v2) \mid v1, v2 \in E(c) \wedge type(v1) = type(v2)\}$$

3. Each subset of the existential variable pairs except the empty set represents a different unification set.

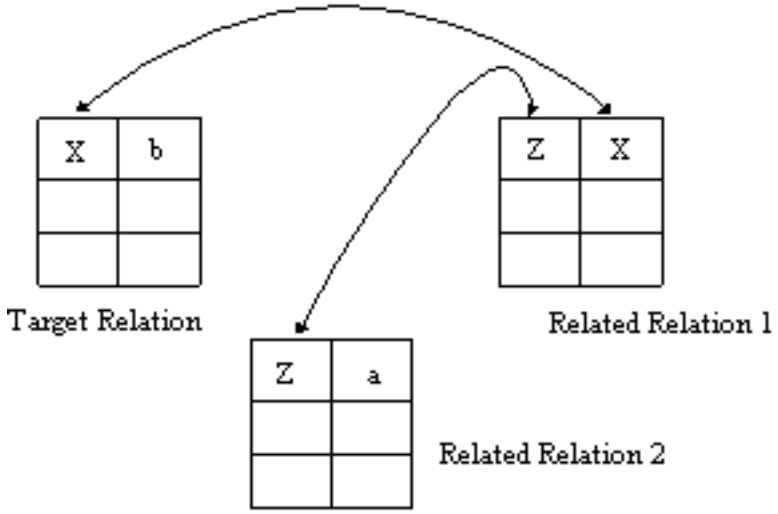

Figure 3.3: A Clause with body predicates indirectly bound to the head predicate

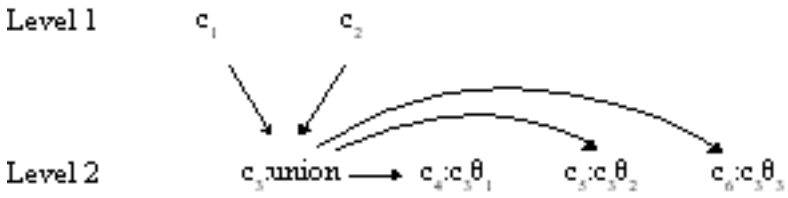

Figure 3.4: The structure of the search space in the Proposed Systems

$$U(c) = \{s \subseteq P(c) \mid s \neq \{\}\}$$

4. For each unification set, a specialization of the union clause is obtained via unifying the variable pairs in the set.

Relational upgrade of APRIORI trick indicates that "If a clause is not frequent then any of its specializations can not be frequent". In other words, the system should not expand any infrequent clause. In the proposed system, there are two specialization operators: adding literals via union operator and unification of existential variables. Therefore, the structure of the search space differs from the APRIORI lattice as in Figure 3.4.

The arrows between different levels represent the union operator and the curves represent the specializations of unions through existential variable uni-

fications. The frequency of union clause c3 depends on the frequency of each clause in the previous level, from which the union is generated, c1, c2; and the specializations in the same level c4, c5, c6 depend on the frequency of union c3. Therefore if the union is not frequent, then the system neither expands the clause nor computes its specializations.

Now, the question is "Does each candidate clause generated represent a meaningful concept description?". The answer is no since there are many concept descriptions in the first level consisting of unbound relations and the union of these clauses may also have unbound relations or partially connected structure. For instance, the clause

$$daughter(X3, ann) : -parent(ann, X3), parent(X28, tom).$$

is refined from the clauses

$$daughter(X3, ann) : -parent(ann, X3).$$

$$daughter(X26, ann) : -parent(X28, tom).$$

The clause has an unbound predicate "parent(X28, tom)" which spoils the relational integrity of the concept description.

If we filter these clauses before adding them to the search space, is there a possibility that the system misses a fully connected rule that is a superset of a partially connected clause? Fortunately, the system does not miss any such clause since a fully connected clause of length i has at least two fully connected disjoint sub-clauses of length i-1, that can be joined into the original clause. Therefore, before adding to the search space each clause should be tested whether it is fully connected or not, except in the first level.

The system utilizes a graph-based method to decide whether a clause is fully connected or not [50]. In this method, first-order clauses are interpreted using a graphical representation. The system checks whether the graph of the clause is connected or not in order to decide the partially connectedness of the clause.

Each predicate of the clause is represented by a vertex and each predicate pair related by shared variables is represented by an indirected arc. For example, the

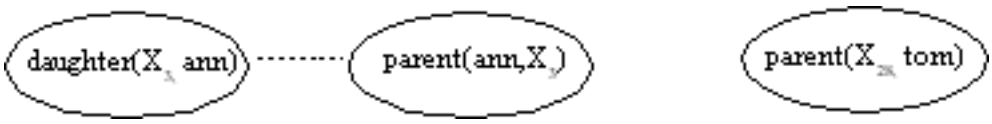

Figure 3.5: A partially connected clause

clause $daughter(X3, ann) : -parent(ann, X3), parent(X28, tom)$ is represented by the graph in the Figure 3.5.

An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph [51]. If the graph of a clause is not connected, i.e. it consists of connected subgraphs, each pair of which has no vertex in common, then the clause is partially connected as in Figure 3.5.

The candidate generation task is given as a pseudo code in Table 3.5.

Table 3.5: The pseudo code of the refinement operator in Proposed System I

For each pair of the clauses k and m in the previous tree level l-1,
do the following:
a.If clause$_k$ and clause$_m$ have same group number, continue.
b.If clause$_k$ and clause$_m$ are both recursive, continue.
c.Compute the union clauses of clause$_k$ and clause$_m$.
d.For each possible union,
i.If tree(level) does not contain the union and the
frequency of the union is above the support threshold, then
1.If the union is a fully connected clause,
add it to the level l; otherwise discard it.
2.Generate clauses by unifying existential variables in
the body clause.
3.For each clause generated, check whether it is frequent
and connected. If it is qualified, add it to the level l.
ii.Else continue.

The proposed refinement operator can be summarized as follows:

Refinement operator Given the language bias L with type declarations, the refinement operator p maps two input clauses c1 and c2, which have all predicates but one body predicate in common, to a set of clauses p(c1, c2) that are the frequent specializations of the union of the input clauses (c1

U c2) under θ-subsumption. In the formal definition,

$$p(c1, c2) = \{c' \in L \mid support(c') \geq threshold,\ (c1 \cup c2)\,\theta = c'\}$$

where θ only unifies one occurrence existential variables in $(c1 \cup c2)$.

For the daughter example, the system selected the concept instance "daughter(mary, ann)", computed the two predicate generalizations of this concept instance in the presence of related facts and populated the first level of the search lattice with these generalizations. It is time to refine the candidate clauses in the first level using the proposed refinement operator.

The system first computes the frequent ones among two predicate candidate clauses via evaluating the support value of each candidate. With the support threshold value 0.8 given through the user interface, the system selects the clauses

$$\begin{array}{ll}
d(X1, X4) : -p(X5, mary). & d(X3, X6) : -p(X6, X3). \\
d(X3, X4) : -p(X5, X3). & d(X1, X4) : -p(ann, X2). \\
d(X1, X6) : -p(X6, X2). & d(X1, X4) : -p(X5, X2). \\
d(X24, X25) : -p(X26, tom). & d(X34, X35) : -f(X34). \\
d(X32, X35) : -f(X33). &
\end{array}$$

with support value 1.0 as frequent and discard the rest of candidates from the first level of the search tree. (d: daughter, p: parent, f: female)

After the frequent clause selection, the candidate clauses for the next level are generated. Lets take the following two frequent clauses in the first level:

$$C1 : daughter(X3, X6) : -parent(X6, X3).$$

$$C2 : daughter(X34, X35) : -female(X34).$$

First, the system checks whether these clauses are appropriate for combining. It checks:

1. Whether the head literals of clauses are equal;

   daughter(X3, X6) and daughter(X34, X35) are equal in predicate level.

2. Whether the clauses have all but one literal in common;

   1-element subsets of first clause are "daughter(X3, X6)" and "parent(X6, X3))"; the second clauses subsets are "daughter(X34, X35)" and "female(X34)". Among these subsets, the clauses "daughter(X3, X6)" and "daughter(X34,X35)" are equal through variable renaming. Therefore, the clauses have 1-element literal set in common.

3. And finally, whether they are specializations/generalizations of the same clause;

   The system keeps a group number for each clause in the search lattice; the specializations of a clause in the same level of the search lattice or generalizations of the same concept instance-related fact pair have the same group number. C1 is a generalization of the clause $daughter(mary, ann)$:- $parent(ann, mary)$ and it has the group number 1; C2 is a generalization of the clause $daughter(mary, ann)$:- $female(mary)$ and has the group number 3. Thereafter, the clauses are not generalizations of the same clause and the combination of them does not cause a redundancy.

Since the clauses are suitable, the system computes the combination of them via the union operator.

1. First the system computes the intersection and the unifying variable renaming θ as in the second check and results in:

$$C_{12} = daughter(X3, X6) \quad \theta : \{X34/X3, X35/X6\}$$

2. Then, the system applies the unifying substitution to the second clause:

$$C_2' = daughter(X34, X35) : -female(X34)\{X34/X3, X35/X6\}$$

$$C_2' = daughter(X3, X6) : -female(X3)$$

3. Then, compute the literal in C_2' but not in C_{12}:

$$C_2' - C_{12} = \{: -female(X3)\}$$

4. Then append this literal to the body of the first clause:

$$C1 \cup C2 = \{daughter(X3, X6) : -parent(X6, X3), female(X3)\}$$

If the support of the union is above the threshold value and does not have partial relations, it will be added to the second level of the search lattice.

After applying the union operator, the system further computes the specializations of the frequent union clauses via unifying existential variables in the body. Since there is no existential variable in "daughter(X3,X6):-parent(X6,X3), female(X3)", the unification will be illustrated on a different clause. For instance, the system applies the union operator to the frequent generalizations {C1, C2} and obtains the union C3 below. Now, we will show how the system unifies the existential variables in the body of the union C3:

$$C1 : daughter(X1, ann) : -parent(ann, X2).$$

$$C2 : daughter(X34, ann) : -female(X35).$$

$$C3 : daughter(X1, ann) : -parent(ann, X2), female(X35).$$

First the system discovers the set of existential variables in the body with its locations, {X2[1,1], X35[2,0]} in the example. Since X2 and X35 are of the "person" type, the set of existential variable pairs of the same type is {(X2[1,1], X35[2,0])}. There is only one subset of existential variable pair set, which is {(X2[1,1], X35[2,0])}, so there is only one specialization of the clause C3 as the variable X35 is renamed as X2:

$$C3' : daughter(X1, ann) : -parent(ann, X2), female(X2).$$

3.2.3 Third Phase: Evaluation of Frequent Clauses

For the first instance of the target concept, which has not been covered by the hypothesis yet, the system constructs the search tree consisting of the frequent

candidate clauses that induces the current concept instance. The system should decide on which clause in the search tree represents a better concept description than other candidates. In other words, the system searches which clause is the best and is suitable to be added to the hypothesis.

Two criteria are important in the evaluation of a candidate concept rule: how much part of the concept instances are captured by the rule and the proportion of the objects which truly belong to the target concept among all those that show the pattern of the rule; support and the confidence, respectively. Therefore, the system should assign a score to each candidate clause according to its support and confidence value.

In the evaluation phase, the system utilizes a metric, *f-measure* [49], based on the support and confidence to evaluate the quality of a clause. The concept description with the highest f-measure value is added to the hypothesis. f-measure can be formally defined as $(2 \times support \times confidence)$ / $(support + confidence)$.

Confidence = Count3 / Count4
where count3 is found as
SELECT count(distinct *) FROM SELECT key fields in the
    body predicates that are bound in the head predicate
    FROM relations in C
    WHERE conditions expressed in the clause C

and count4 is found as
SELECT count(distinct *) FROM SELECT key fields in the body
    predicates that are bound in the head predicate
    FROM relations in the body of C
    WHERE conditions expressed in the body of C

The computation of the support of a definite clause is described in Section 3.2.2.1. The *confidence* of a definite clause C, the probability of the key objects

that satisfy the body literals of the clause also satisfies the head, can be obtained with the previous formula via SQL queries.

For instance, for clause daughter(X, Y):- parent(Y, X), the following query computes the confidence value in SQL syntax:

```
Confidence = Count5 / Count6
where count5 is found as
SELECT count(distinct *) FROM
        SELECT parent.arg1, parent.arg2
        FROM daughter, parent
        WHERE daughter.arg1 = parent.arg2
                AND daughter.arg2 = parent.arg1
and count6 is found as
SELECT count(distinct *) FROM
        SELECT parent.arg1, parent.arg2
        FROM parent
```

According to the daughter database, this query returns the confidence value of 2/3.

In the daughter example, the search tree constructed for the daughter instance "daughter(mary, ann)" is traversed for the best clause and the clause "daughter(X3,X6):-parent(X6,X3), female(X3)" with support value of 1.0 and the confidence value of 1.0 is selected and added to the hypothesis. Since all the concept instances are covered by this rule, the algorithm terminates and outputs the hypothesis:

$$daughter(X3, X6) : -parent(X6, X3), female(X3)$$

3.3 Proposed System II

In order to capture clauses that have relations not directly bound to the head predicate, the proposed system allows fully existential/unbound predicates in

Table 3.6: The Proposed System I

- Initialize the set of concept instances set I
- Initialize the hypothesis H = $\emptyset$
- Do until all the concept instances are covered by the hypothesis (I = $\emptyset$):
    1.Select the first positive concept instance p from I.
    2.Generalize positive instance in presence of background knowledge via Algorithm 2 and call the set of generalizations G.
    3.Initialize level d := 1
    4.Initialize the set of candidate clauses C_1 := G
    5.Initialize the set of frequent queries F := {}
    6.While Q_d not empty and d $\leq$ *maxdepth*
        a. Find frequency of all clauses C $\in$ C_d
        b. Discard the clauses with frequency below *minfreq* from C_d.
        c. Update F := F $\cup$ C_d
        d. Compute new candidates Q_{d+1} from $_{d+1}$, using Algorithm 3
        e. Increment d by 1.
    7.Discard the clauses with confidence below *minconf* from F.
    8. Select the best clause cbest from F using the f-measure criterion.
    9.Compute the set of concept instances I_c covered by the best clause.
    10.Update H := H $\cup$ c_{best}
    11.Update I := I - I_{bc}
- Return H.

the body of clauses in the generalization step. Therefore, the first level of the search lattice expands exponentially as the number of facts related to the current concept instance increases. Since the size of each level l of the ARPIORI search lattice is order of two squared the size of the level l-1, the size of the search lattice is order of $n^{2^{(d-1)}}$ in the worst case, where n is the size of the first level and d is the depth of the search tree.

For large scale data mining tasks like discovering structure-activity relationships (SAR) that relate molecular structure with specific ability of molecules, the background knowledge database is generally composed of 20000 records or more, which results in an intractable problem for the proposed system I. There is a tradeoff between the complexity and the completeness of the algorithm. In our less complex and then less complete solution, the system tightens the limits of the language bias in the sake of efficiency. The proposed system II does not

allow clauses with body relations not directly bound to the head predicate in the language.

We change the generalization and refinement operators of the proposed system I in the second version.

The concept learning time is determined by the number of two predicate clauses in the first level of the search space. Therefore, the generalization operator of the proposed system II differentiates in the way that it does not allow two predicate clauses that have body predicates not bound to the head.

Besides the limitation on the body predicates, the generalization operator selectively substitutes one-location existential variables (variable that is not bound in the head and exists only once) to terms in the body predicates.

In fact, assigning one-location existential variable to an argument location means ignoring that attribute of the predicate in describing the concept pattern. As in [52], the predicates in the database can be classified as "utility predicates" and "structural predicates". *Structural predicates* explain the substructures of an object; whereas *utility predicates* present the properties of objects. For instance, for a molecule, "has_atom(moleculeId, atomId)" is a structural predicate, while "mutagenic(moleculeId)" is a utility one. In a concept description, we can only ignore features of an object or extra properties of a structural relation. The object itself or a relation should not be ignored if that object or relation is added to the rule. Therefore, the system imposes a language bias such that key arguments of predicates can not be assigned one-location existential variables.

However, if there is a one-to-many relation between key fields of a predicate and the predicate encapsulates the properties of the key field with many arity, then it is necessary to assign one-location existential variables to key fields. This kind of predicates is a combination of utility and structural predicates and should be normalized into structural and utility predicates separately. This in fact introduces a new pre-processing step before the learning task. In the test phase, key fields with many arity are ignored and the system does not assign one-location existential variable to any key fields of a body predicate.

As stated in [26], there are literals which do affect neither the support nor the confidence of a relational association rule if added. We will call these predicates as *valid predicates*. For instance, the target concept is "class(animal, type)" and the background knowledge contains "habitat(animal, location)" and "has_eggs(animal)". The predicate "has_eggs(animal)" is not true for all animals and is a distinctive property among animals. On the other hand, the fact "habitat" returns always true since every animal has a habitat. Therefore, the literal "habitat(X, Y)" does not distinguish one animal class from another if added to the body of a rule as a new condition. In a formal way, valid predicates can be defined as follows:

$\forall k_1, k_2, ..., k_n \Rightarrow \exists a_1, a_2, ..., a_m \quad predicate\ (k_1, k_2, ...k_n, a_1, a_2, ..., a_m) \quad where\ k_i \in keys$ and $a_j \in attributes.$

We state that it is not sound to make such literals fully existential in the generalization step via assigning existential variable to all attributes of the literal. This restriction prevents the potential redundancy in the lower levels of the search space.

This language bias is applied to only body predicates and local variables in the body; there is no limitation in assigning existential variables to the head variables since existential variables in the head are necessary for clauses to be combined in the specialization steps.

The user declares such literals (always true or not) and key fields of each background predicate via graphical interface.

In the daughter example, the following two predicate clauses, whose body predicates are bound to the head, are generalized from the concept instance "daughter(mary, ann)" and the related fact "parent(ann, mary)" in the generalization phase:

Since the arguments of both "daughter" and "parent" relations are all key fields for the corresponding relations, the system does not assign existential variable to any fields of body predicates. In comparison with the generalization operator in the first implementation, the number of rules in the first level of the

Table 3.7: Two predicate concept descriptions generated by the Proposed System II

Concept Descriptions	**Value of θ_2^{-1}**
daughter(X3,ann):-parent(ann,X3)	mary-X3-[0, 0][1, 1]
daughter(mary,X6):-parent(X6,mary)	ann-X6-[0, 1][1, 0]
daughter(X1,X6):-parent(X6,mary)	mary-X1-[0, 0], ann-X6-[0, 1][1,0]
daughter(X3,X4):-parent(ann,X3)	mary-X3-[0, 0][1, 1], ann-X4-[0,1]
daughter(X3,X6):-parent(X6,X3)	mary-X3-[0, 0][1, 1], ann-X6-[0, 1][1,0]

search tree reduces significantly from 18 to 5 clauses.

After the generalization step, the rules in the last level of the search tree are combined in order to retrieve candidate rules of the current level.

In the Proposed System I, an extra specialization step is employed, via unifying the existential variables, to capture inner structures not directly related to the head predicate, in other words n-depth relational patterns. In this implementation, the system does not allow fully existential body predicates in the generalization step. Therefore, an extra generalization step is employed after combining the clauses in order to discover the first-order features, which are sets of body literals interacting by local variables [52]. In the generalization step, new local variables are introduced by unifying common constants of the same type in various argument positions of body predicates.

The generalization after specialization constitutes a breach in the application of the APRIORI rule since any generalization of the specialization of two clauses can be more general than one of or both of the clauses. Therefore, this prevents the top-down specialization of the APRIORI lattice. As a result, it is possible to bypass some frequent definite clauses because of this gap.

Finally, the filtering step, which checks whether the candidate clause is connected or not, is not employed in this version since the clauses are guaranteed to be connected via head variables.

In the daughter example, the following frequent two predicate generalization set is obtained as a result of the generalization of the concept instance

daughter(mary, ann), in the presence of related facts, in this implementation.

$$\{daughter(X2, X1) : -female(X2)., daughter(X8, X6) : -parent(X6, X8).\}$$

The generalizations are appropriate for combining and the resultant clause is as follows:

$$daughter(X2, X1) : -female(X2), parent(X1, X2).$$

Since the uncommon literals of two clauses, female(X2) and parent(X1, X2), do not have common constants that can be variablized, there is no generalizations of this clause. Therefore, the second level of the search space only consists of the clause "$daughter(X2, X1) : -female(X2), parent(X1, X2)$." and the algorithm terminates.

3.4 Discussion

There is a trade-off between two versions of the proposed technique. The proposed system I allows the clauses including body predicates that are not directly connected to the head predicate in the search space, whereas the second one does not in the sake of efficiency. The efficiency of the second system lies in not allowing fully existential body predicates in the generalization step. Then the question arises, "Is not it possible to construct a combination of these systems that does not allow fully existential body predicates in the first phase and achieves to cover clauses with body predicates not directly connected to the head?" It is only possible by adding literals to the body of the clauses in inner steps and bounding these literals to only body predicates. This extra specialization step also results in performance overhead as introducing existential variables in the first level.

CHAPTER 4

EXPERIMENTAL RESULTS

The experiments show how the first system can be used to learn a linear recursive relation, and second one to predict carcinogenicity of a molecule in a large noisy database. Additionally, two systems are compared on a benchmark ILP problem, the task of learning finite element mesh design. Each experiment is performed on a desktop computer equipped with 256 MB RAM and 1.80 GHz Pentium 4 processor.

4.1 Linear Recursive Rule Learning

The first test case is a complex family relation, "same-generation", learning problem. In the data set, 118 pairs of family members are given as positive examples of "same-generation" relation. Additionally, 30 background facts are provided to describe the parental relationships in the family. The task is to form general rules to describe the "same-generation" concept.

Types of predicate arguments are provided by the user via graphical interface and stored in a source file. Positive concept instances and background knowledge are given as DB2 database tables. The following items show the data and user declarations utilized by the program:

- Types

parent(person, person).

same-generation(person, person).

- Concept instances: Same-generation relation

Person1	Person2
Seda	Eda
Eda	Seda
...	...

- Background knowledge: Parent relation

Parent	Child
Adnan	Seda
Adnan	Eda
...	...

The parameters of the algorithm, support and confidence values are set to 0.3 and 0.6, respectively. Assuming that the concept description will contain n disjunctive rules and the concept instances are shared between the rules homogenously, then the minimum support value should be 1/n. Since we assume that the hypothesis has 3 rules, the support threshold is set to 0.3. One more parameter, the maximum predicate count of the clause, is set to 5. With these parameters set, the proposed system I found the following hypothesis in 8 seconds: (s: samegeneration, p: parent)

s(X2,X1):-s(X3,X5), p(X5,X2),p(X3,X1).
s(X2,X1):-s(X4,X3), p(X4,X2),p(X3,X1).
s(X2,X1):-p(X5,X2), p(X5,X1).

The first two clauses show that "same generation" relation is a symmetric relation and the third relation forms the base clause of the recursive relation.

Since the second proposed system can not find clauses with body predicates indirectly bound to the head, the first two recursive relations will not be captured

in the hypothesis if this system tries to learn the "same-generation" relation. As a result, second version is not capable of learning "same-generation" concept.

4.2 The Carcinogenicity Evaluation Problem

A large percentage of cancer incidents stems from the environmental factors, such as cancerogenic compounds. The carcinogenicity tests of compounds are vital to prevent cancers; however, the standard bioassays of chemicals on rodents are really time-consuming and expensive. Therefore, the National Toxicity Program (NTP) of the U.S. National Institute for Environmental Health Sciences (NIEHS) started the Predictive Toxicology Evaluation (PTE) project in order to relate the carcinogenic effects of chemicals on humans to their substructures and properties using the machine learning methods [53].

In the NTP program, the tests conducted on the rodents results in a database of more than 300 compounds classified as carcinogenic or non-carcinogenic. Among these compounds, 298 of them are separated as training set, 39 of them formed the test set of the first PTE challenge (PTE-1) and the other 30 chemicals constitute the test set of the second PTE challenge (PTE-2) for the data mining programs [31].

Within these challenges, knowledge discovery algorithms are presented with characteristics of molecules that may be associated with carcinogenesis as background knowledge having roughly 25,500 facts [54]:

- Atom-bond structures of molecules.
- Complex structural groups like methyl, benzene.
- Inducing potential genetic risks determined as a result of short-term bioassays.
- Being mutagenic or not.
- Having some structural alerts.

In this experiment, we tested the second implementation against the PTE-1 test set using the data published in [55].

The test procedure can be decomposed in two main phases: Parameter setting and model construction from the training data, testing the model on the test set. We used 298 molecules labelled as "training set" in order to obtain hypotheses and PTE-1 test set in order to evaluate the constructed hypotheses.

There are three important parameters in the proposed system: minimum support value (support threshold), minimum confidence value (confidence threshold) and the maximum number of predicates in the clause. The system should decide the optimum values of these parameters for the carcinogenesis domain.

The method for choosing the optimum values for the parameters is simple:

1. Change the values of the parameters as one parameter varies at a time.
2. With this parameter setting, obtain hypotheses describing the concept from the training set.
3. Evaluate and record the score of the hypotheses on the test set.
4. Choose the parameter setting and the hypotheses that result in the most accurate prediction.

In hypothesis generation phase, our technique defines both the carcinogenicity and non-carcinogenicity instead of modelling only the carcinogenicity concept, since labelling the chemicals as negative in carcinogenic activity that do not show the patterns of carcinogenicity is not logical. This is due to the fact that the database may not include all the patterns of the carcinogenicity and some molecules may be labelled as "unknown".

The evaluation criteria of hypotheses employed is predictive accuracy. For each molecule in PTE-1 test set, the rules in both hypotheses are scanned to check whether they cover the current concept instance. If the current concept instance does not have any patterns introduced by rules of the hypotheses, then it is labelled as "unknown". Otherwise, the one having the highest f-measure

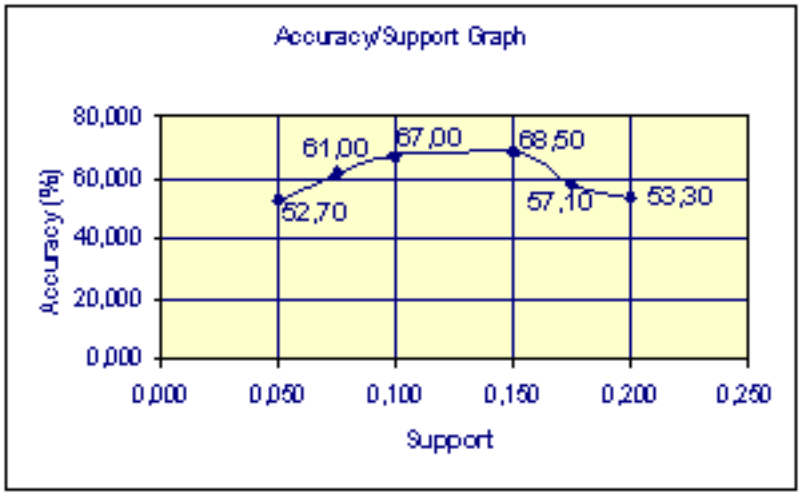

Figure 4.1: Predictive Accuracy/Support Graph

value among the rules covering the current concept instance determines the class (carcinogenic or non-carcinogenic) of the molecule. After evaluating each molecule in the PTE-1 test set, the *predictive accuracy* of the hypothesis is computed by the proportion of the sum of positive concept instances classified as positive (true positives) and negative concept instances classified as negative (true negatives) to the total number of the concept instances that the hypothesis classifies. The total number of concept instances classified by the hypothesis is also called the *coverage* of the hypothesis. [31].

First, we trained the system with varying values of the support threshold while the confidence threshold and the maximum number of predicates are fixed to 0.6 and 5, respectively. The support values and the corresponding predictive accuracies of the resultant hypotheses are plotted in Figure 4.1.

The graph shows that a very low or very high support threshold results in low accuracy. If the minimum support value is too high (> 0.15), then the rules involving patterns that rarely occur are not generated. Besides, rules that partition the concept instances into many small subsets may be generated, many of which are not correct if it is set too low (< 0.1). Therefore, the minimum support value in the range of [0.1, 0.15] should be preferred.

After selecting the optimum value 0.15 for the support value, we obtained

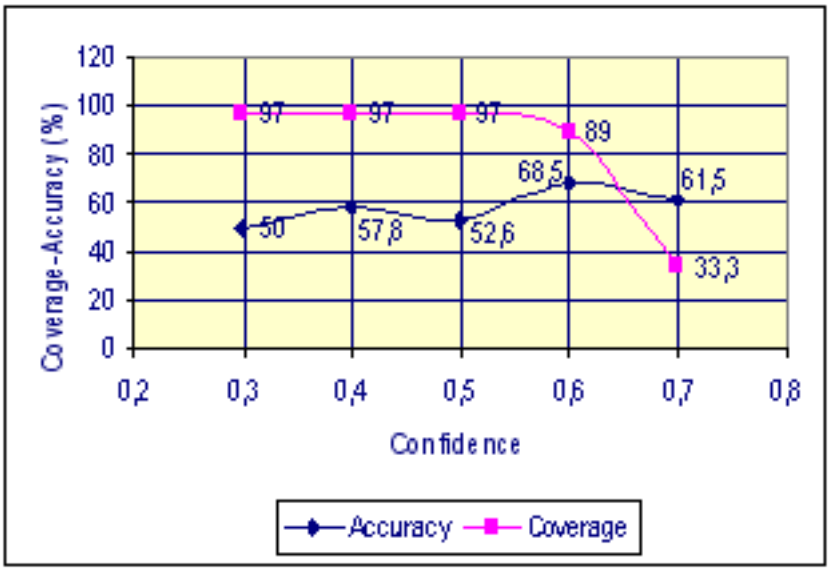

Figure 4.2: Predictive Accuracy/Confidence Graph

and tested theories for different confidence values in the range [0.3, 0.8]. The test results are summarized in Figure 4.2.

Normally, the rules having 100% confidence value should appear in the final hypothesis; however, there can be noise in the data that should be tolerated.

There is no regular behavior of the accuracy in confidence threshold values below 0.5 and this tells us that confidence below a minimum value is not a criterion in discriminating performance of the rules. However, accuracy steadily decreases as the confidence increases from the value of 0.6. This can be explained by the decrease in the coverage of the hypothesis.

As can be seen from the graph, the confidence threshold value 0.6 results in the best predictive accuracy. To set the minimum confidence to 0.6 means that the system tolerates 40% noise at maximum. However, 40% noise for the carcinogenesis data is an aggregated value since the data is obtained from the long-lasting bioassays. We can conclude that not the noise but the missing data about molecular structures and properties in the background knowledge pulls down the minimum confidence value.

After fixing the support and confidence values, we test the system with dif-

Table 4.1: The carcinogenicity hypothesis of the Proposed System II

A compound is carcinogenic if
1. It tests positive in Ames biological test; or
2. It has a "halide10" group recognized by Ashby and an aromatic halide group; or
3. It has a methyl group and a positive cytogen_ca test; or
4. It has a positive cytogen_ca test and cytogen_sce test; or
5. It has a negative "chromosomal aberration" test; or
6. It has a "amino" and "di10" group recognized by Ashby and an amine group; or
7. It has a positive salmonella test; or
8. It reacts the same against both salmonella and cytogen_ca tests, it tests positive in cytogen_sce test and has an aromatic halide group; or
9. It has positive mouse_lymph and cytogen_ca tests.

ferent values of maximum clause length in the range of [6, 10]. The accuracy and the final hypothesis is the same for all values, in other words the system does not find any rule having more than 6 predicates. Therefore, maximum clause length is set to 6 due to the efficiency.

With the optimum parameter settings, the hypotheses constructed by the proposed system are illustrated in Table 4.1 and Table 4.2. The proposed system II generated the hypotheses for the carcinogenicity and non-carcinogenicity in 10 and 15 minutes, respectively. The predictive accuracy of 68.5% is achieved with these concept models for PTE-1 test set.

The patterns in the hypothesis of carcinogenesis, involve only results of biological tests and the presence of chemical compounds; however, there is no rule about the molecular substructures composing of atom and bond relations. Since there are one-to-many relations among the keys of "atom" and "bond" relations (a molecule can have more than one atom and bond), one-location existential variables should be assigned to these key fields, which is forbidden due to the language bias. Therefore, the rules like "molecules having bromine element are carcinogenic" can not be expressed since it is not possible to assign one-location existential variable to the "AtomID" key field, as in "atm(MolID, AtomID, br,

Table 4.2: The non-carcinogenicity hypothesis of the Proposed System II

A compound is non-carcinogenic if
1. It has a six-ring group, tests negative in salmonella test and has a positive mouse_lymph test; or
2. It has a six-ring group, tests negative in salmonella test and has a positive mouse_lymph test; or
3. It has negative salmonella and cytogen_ca tests, and it is mutagenic; or
4. It has methyl and six_ring groups, and has negative salmonella and cytogen_ca tests; or
5. It has amine and six_ring groups, and it is mutagenic; or
6. It has a negative cytogen_ca test, it is mutagenic and it has a ketone group; or
7. It has an amine group, and it has negative salmonella and cytogen_ca tests; or
8. It tests negative in salmonella and cytogen_ca tests and has a positive mouse_lymph test; or
9. It tests negative in cytogen_ca test, it has a six-ring group and it has a positive mouse_lymph test; or
10. It is mutagenic, it has a positive mouse_lymph test and has six-ring and non_ar_6c_ring groups; or
11. It has a negative salmonella test and has six-ring, methyl, ether and non_ar_6c_ring groups.

ElementCount, AtomCharge)".

When we analyze the PROGOL's hypothesis in [54], we come up with the rules 1, 5 and 6 in Table 4.1. The PROGOL's hypothesis is composed of nine first-order clauses and five of the clauses have patterns related to the atom-bond relations in the molecule. From this point, we can conclude that the predictive accuracy of the proposed algorithm increases if carcinogenesis data is normalized in a preprocessing step and also continuous data (atom charge) is handled.

The PTE-1 predictive accuracies of the state-of-art methods are listed in Table 4.3 as in [54].

4.3 Finite Element Mesh Design

In mechanical engineering, physical structures are represented by finite number (mesh) of elements to sufficiently minimize the errors in the calculated deforma-

Table 4.3: The predictive accuracies of the state-of-the-art methods for the carcinogenicity problem

Method	**Type**	**Predictive Accuracy**
Ashby	Chemist	0.77
PROGOL	ILP	0.72
RASH	Biological potency analysis	0.72
Proposed Algorithm	**ILP + Data mining**	**0.685**
TIPT	Propositional ML	0.67
Bakale	Chemical reactivity analysis	0.63
Benigni	Expert-guided regression	0.62
DEREK	Expert System	0.57
TOPCAT	Statistical Discrimination	0.54
COMPACT	Molecular Modeling	0.54
Default	Majority Class	0.51

tion values. The problem is to determine an appropriate mesh resolution for a given structure, that results in accurate deformation values.

Mesh design is in fact determination of the number of elements on each edge of the mesh. The task is to learn the rules to determine the number of elements for a given edge in the presence of the background knowledge such as the type of edges, boundary conditions, loadings and the geometric position.

Four different physical structures called (a-e) in [56] are used for learning in this experiment. The number of elements on each edge in these structures are given as positive concept instances, in the form of mesh(Edge, NumberOfElements). An instance of the positive example, mesh(b1, 6), means that edge 1 of the structure b should be divided in 6 sub-edges. The background knowledge contains information about edge types, loadings on the edges, boundary conditions and relative geometric positions of the edges. Some of background facts are as follows [57]:

Edge Types:	long, circuit, short-for-hole
Boundary Conditions:	fixed, free, two-side-fixed
Loadings:	not-loaded, one-side-loaded, cont-loaded
Geometrical Relations:	neighbor-xy, opposite, same

There are 278 positive concept examples and 1872 background facts in the data set. We ran both versions of the proposed technique on this data set by deriving a set of definite clauses for the four structures (b, c, d, e) and testing these clauses on the remaining structure (a). The classification phase evaluates the induced hypothesis according to the number of the edges in the test structure that are assigned the correct number of elements (correctly classified).

We set the support threshold to 0.2, the confidence threshold to 0.6 and the maximum number of predicates in the clause to 5 for both versions. Additionally, another restriction is applied on the final rules to be induced such that there can be at most two structural predicates neighbor, opposite, equal in a rule due to the tractability issues.

The first version of the proposed system ran on this data for approximately 1.5 hours and generated 36 rules that describe the edges with 1 to 12 number of elements. Some of the rules are given in Prolog format in Table 4.4. The hypothesis correctly classifies 18 of the 55 edges in the structure a.

Before we test the second proposed system, we determined the key fields of the relations (the fields of type edge are the key fields) and the relations that always return true (only "mesh" relation) via user interface. The second system was run on the same data-set for 5 minutes and it generated 13 rules. Some of the rules in the second hypothesis are illustrated in Table 4.5. The classification accuracy of the second hypothesis is lower than the first one and only 7 of the 55 edges are correctly classified.

As it can be seen in Table 4.4, first proposed system found rules that include the properties of the related edges via body predicates not directly bound to the head. However, second system induce rules that use only the properties of a given edge, no related edges. This fact reflects on the predictive accuracies of

Table 4.4: Some of the rules in the hypothesis of the Proposed System I for the mesh problem

mesh(X2,1):- neighbor_yz(X1,X2), not_important(X2).
mesh(X7,2):- neighbor_xy(X7,X8), short(X7).
mesh(X8,7):- neighbor_xy(X8,X9), fixed(X9), not_loaded(X9), long(X8).
mesh(X8,7):- opposite(X8,X5), mesh(X5,7), fixed(X5), not_loaded(X8).

Table 4.5: Some of the rules in the hypothesis of the Proposed System II for the mesh problem

mesh(X2,1):- not_loaded(X2), not_important(X2).
mesh(X6,1):- free(X6), not_important(X6).
mesh(X4,12):- circuit(X4), not_loaded(X4).
mesh(X1,12):- circuit(X1), free(X1).

the systems. On the other hand, second system discovered the rules 18 times faster than the first system.

The predictive accuracies of the state-of-art methods listed in [57] show that FOIL, GOLEM, MFOIL and CLAUDIEN correctly classify 17, 17, 22 and 31 of 55 edges in the structure a, respectively. The classification accuracies of the systems are not very high. This is due to the sparseness of the data and each of the five structures shows unique characteristics that can not be captured in the rules [2].

CHAPTER 5

CONCLUSION

In this study, many aspects of multi-relational data mining are examined and discussed. The aim is to combine rule extraction methods in ILP and efficient search strategies of data mining. As an outcome, two versions of a concept learning tool, a modified combination of WARMR and Inverse Resolution absorption operator, is produced. The application area of the first implementation is learning complex, mostly recursive concepts/relations. Besides, the second limited implementation is suitable for discovering frequent patterns in large and noisy data sets, in other words, performance-critic applications.

Both versions of the proposed system are tested on popular data mining search areas, such as bioinformatics and mesh design, and we come up with promising test results that are comparable with the performance of current state-of-the-art knowledge discovery systems, such as PROGOL.

Additionally, this study introduces a new method that induces modes of predicate arguments, referenced or non-referenced, via inputting basic domain knowledge from the user. The non-referenced arguments are in fact one-location existential variables; an n-argument predicate results in 2^n different combinations of instantiations if non-referenced variables are allowed for all argument positions of the predicate, resulting in exponential grow of the search space. The proposed mode induction algorithm speeds up the learning process, by determin-

ing which predicate arguments must be referenced in the clause or which ones can be ignored, not referenced. However, the methodology requires normalized data set in which utility and structural predicates are isolated. Real world databases are usually not normalized and require an embedded pre-processing normalizing step. This normalizing step is another research area, and not employed in the learning phase of the proposed systems.

One limitation of the proposed system is that it only handles categorical data and can not benefit from numerical fields, like atom charge in the carcinogenesis test, in order to extract more strong rules. Besides, it inputs only ground background facts and can not process background rules about domain.

The results for the prediction of carcinogenesis and mesh design show that the combination of the merits of ILP and data mining areas result in a promising knowledge discovery system.

REFERENCES

[1] S. Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations*, 5(1):1–16, 2003.

[2] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.

[3] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.

[4] G. Cestnik, I. Konenenko, and I. Bratko. Assistant-86: A knowledge-elicitation tool for sophisticated users, in Progress in Machine Learning, I. Bratko and N. Lavrac, Eds. Wilmslow, U.K.: Sigma, 1987, pp. 31–45.

[5] I. Mozetic. NEWGEM: Program for learning from examples. Technical documentation and user's guide. Reports of Intelligent Systems Group UIUCDCS-F-85-949, Department of Computer Science, University of Illinois, Urbana Champaign, IL, 1985.

[6] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proc. Fifth European Working Session on Learning*, pages 151–163, Berlin, 1991. Springer.

[7] S. Muggleton. Inductive Logic Programming. In *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. MIT Press, 1999.

[8] S. Kramer and G. Widmer. Inducing classification and regression trees in first order logic. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 140–159. Springer-Verlag, September 2001.

[9] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[10] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, 1999.

[11] W. Emde and D. Wettschereck. Relational instance based learning. In Lorenza Saitta, editor, *Machine Learning - Proceedings 13th International Conference on Machine Learning*, pages 122 – 130. Morgan Kaufmann Publishers, 1996.

[12] M. Kirsten, S. Wrobel, and T. Horvath. Distance based approaches to relational learning and clustering. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 213–232. Springer-Verlag, September 2001.

[13] D.W. Aha. Relating relational learning algorithms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 233–260. Academic Press, 1992.

[14] J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5(3):239–266, 1990.

[15] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrač. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 1041–1047, 1986.

[16] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983.

[17] L. De Raedt and M. Bruynooghe. Constructive induction by analogy. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 476–477. Morgan Kaufmann, 1989.

[18] S. Wrobel. Automatic representation adjustment in an observational discovery system. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 253–262. Pitman, 1988.

[19] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335–359. Academic Press, 1992.

[20] C. Sammut and R. Banerji. Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume 2*, pages 167–191. Morgan Kaufmann, 1986.

[21] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.

[22] S. H. Muggleton. DUCE: An oracle-based approach to constructive induction. In *Proc. Tenth International Joint Conference on Artificial Intelligence*, pages 287–292, San Mateo, CA, 1989. Morgan Kaufmann.

[23] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[24] S. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer-Verlag, 1996.

[25] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[26] L. P. Castillo and S. Wrobel. Macro-operators in multirelational learning: a search-space reduction technique. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proceedings of the 13th European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Artificial Intelligence*, pages 357–368. Springer-Verlag, August 2002.

[27] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[28] P. E. Utgoff and T. M. Mitchell. Acquisition of appropriate bias for inductive concept learning. In *Proceedings of the 1st National Conference on Artificial Intelligence*, pages 414–417, 1982.

[29] J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[30] F. Giannotti, G. Manco, and J. Wijsen. Logical languages for data mining. In *Logics for Emerging Applications of Databases*, pages 325–361, 2003.

[31] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1–6. Morgan-Kaufmann, 1997.

[32] S. Muggleton. Inductive logic programming. *New Gen. Comput.*, 8(4):295–318, 1991.

[33] S. Muggleton, editor. *Inductive logic programming.* Academic Press, London, 1992.

[34] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Three companions for data mining in first order logic. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 105–139. Springer-Verlag, September 2001.

[35] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[36] N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Trans. Comput. Logic*, 2(4):458–494, 2001.

[37] G.D. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, 1971.

[38] Peter Idestam-Almquist. Generalization of clauses under implication. *J. Artif. Intell. Res. (JAIR)*, 3:467–489, 1995.

[39] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[40] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM Press.

[41] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[42] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.

[43] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In *SIGMOD'98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 343–354, New York, NY, USA, 1998. ACM Press.

[44] Jeffrey D. Ullman. CS345 Lecture Notes about Association-Rules, http://www-db.stanford.edu/ullman/cs345-notes.html, Last Updated on 2003.

[45] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[46] L. Dehaspe and H. Toivonen. Discovery of relational association rules. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 189–212. Springer-Verlag, September 2001.

[47] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *ILP'97: Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 125–132, London, UK, 1997. Springer-Verlag.

[48] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*, pages 63–92. Academic Press, 1992.

[49] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. In N. Kasabov and K. Ko, editors, Proc. ICONIP/ANZIIS/ANNES'99 International Workshop: Emerging Knowledge Engineering and Connectionist-Based Information Systems, pages 192-196, Dunedin, NZ, 1999.

[50] An open source graph component, http://www.jgraph.com/.

[51] K. H. Rosen. *Discrete mathematics and its applications*. McGraw-Hill, Inc., New York, NY, USA, 1988.

[52] P. A. Flach and N. Lavrač. The role of feature construction in inductive rule learning. In L. De Raedt and S. Kramer, editors, *Proceedings of the ICML2000 workshop on Attribute-Value and Relational Learning: crossing the boundaries*, pages 1–11. 17th International Conference on Machine Learning, July 2000.

[53] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36. AAAI Press., 1998.

[54] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 273–287. Springer-Verlag, 1997.

[55] Oxford University Computing Laboratory. Predicting carcinogenecity, http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/cancer.html, Last Updated on November 1999.

[56] B. Dolšak and S. Muggleton. The application of inductive logic programming to finite-element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, 1992.

[57] W. Van Laer, L. Dehaspe, and L. De Raedt. Applications of a logical discovery engine. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 263–274, 1994.