AN ILP-BASED CONCEPT DISCOVERY SYSTEM FOR MULTI-RELATIONAL DATA MINING

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

BY

YUSUF KAVURUCU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JULY 2009

Approval of the thesis:

AN ILP-BASED CONCEPT DISCOVERY SYSTEM FOR MULTI-RELATIONAL DATA MINING

submitted by **YUSUF KAVURUCU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit _____
Head of Department, **Computer Engineering**

Asst. Prof. Dr. Pınar Şenkul _____
Supervisor, **Computer Engineering Department, METU**

Examining Committee Members:

Prof. Dr. Özgür Ulusoy
Computer Engineering Dept., Bilkent University _____

Asst.Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU _____

Prof. Dr. Adnan Yazıcı
Computer Engineering Dept., METU _____

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Date: _____

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: YUSUF KAVURUCU

Signature :

ABSTRACT

AN ILP-BASED CONCEPT DISCOVERY SYSTEM FOR MULTI-RELATIONAL DATA MINING

Kavurucu, Yusuf
Ph.D., Department of Computer Engineering
Supervisor : Asst. Prof. Dr. Pınar Şenkul

July 2009, 118 pages

Multi Relational Data Mining has become popular due to the limitations of propositional problem definition in structured domains and the tendency of storing data in relational databases. However, as patterns involve multiple relations, the search space of possible hypothesis becomes intractably complex. In order to cope with this problem, several relational knowledge discovery systems have been developed employing various search strategies, heuristics and language pattern limitations.

In this thesis, Inductive Logic Programming (ILP) based concept discovery is studied and two systems based on a hybrid methodology employing ILP and APRIORI, namely Confidence-based Concept Discovery and Concept Rule Induction System, are proposed.

In Confidence-based Concept Discovery and Concept Rule Induction System, the main aim is to relax the strong declarative biases and user-defined specifications. Moreover, this new method directly works on relational databases. In addition to this, the traditional definition of confidence from relational database perspective is modified to express Closed World Assumption in first-order logic. A new confidence-based pruning method based on the improved

definition is applied in the APRIORI lattice. Moreover, a new hypothesis evaluation criterion is used for expressing the quality of patterns in the search space. In addition to this, in Concept Rule Induction System, the constructed rule quality is further improved by using an improved generalization metod.

Finally, a set of experiments are conducted on real-world problems to evaluate the performance of the proposed method with similar systems in terms of support and confidence.

Keywords: Inductive Logic Programming (ILP), Multi-Relational Data Mining, Concept Discovery, Support, Confidence

ÖZ

ÇOK İLİŞKİLİ VERİ MADENCİLİĞİ İÇİN TÜMEVARAN MANTIKSAL PROGRAMLAMA TABANLI KONSEPT TANIMLAMA SİSTEMİ

Kavurucu, Yusuf

Doktora, Bilgisayar Mühendisliği

Tez Yöneticisi : Y.Doç. Dr. Pınar Şenkul

Temmuz 2009, 118 sayfa

Yapısal kümelerde problem tanımlamalarını tek bir bağıntı ile yapmanın getirdiği sınırlamalar ve verileri ilişkisel veri tabanlarında saklama eğilimine bağlı olarak, Çok İlişkili Veri Madenciliği popüler hale gelmiştir. Bununla birlikte, bilgi örüntüleri birden fazla ilişki içermeye başladıkça, olası örüntü arama uzayı kolay işlenemeyecek kadar büyümüştür. Söz konusu problemi çözmek için, farklı arama stratejileri, varsayımları ve örüntü dil kısıtları kullanan birçok çok-ilişkili bilgi çıkaran sistem geliştirilmiştir.

Bu tezde, Tümevaran Mantık Programlama (TMP) tabanlı konsept bulma sistemleri çalışılmış ve TMP ile APRIORI tekniklerini kullanan bir hibrid metodoloji baz alınarak Doğruluk-tabanlı Konsept Bulma ve Konsept Kural Tümevarım Sistemi isimli iki sistem anlatılmıştır.

Doğruluk-tabanlı Konsept Bulma ve Konsept Kural Tümevarım Sistemi metotlarındaki asıl amaç, katı bildirim kısıtları ve kullanıcı-tanımlı özellikleri gevşetmektir. Ayrıca, bu yeni metod doğrudan ilişkisel veritabanları üzerinde çalışmaktadır. Buna ek olarak, ilişkisel veritabanları açısından yapılan geleneksel doğruluk tanımı, Kapalı Dünya Varsayımını birinci-derece mantıkla açıklamak için modifiye edilmiştir. Geliştirilmiş tanım üzerinde yeni bir

doğruluk-tabanlı budama metodu APRIORI örüsüne uygulanmıştır. Ayrıca, arama uzayındaki kalıpların kalitesini açıklamak için, yeni bir hipotez değerlendirme kriteri kullanılmıştır. Buna ilave olarak, geliştirilmiş genelleme metodu kullanılarak Konsept Kural Tümevarım Sistemi algoritmasında üretilen kural kalitesi iyileştirilmiştir.

Sonuç olarak, önerilen metodun literatürdeki benzer sistemlere oranla performansını doğruluk ve kapsama açısından değerlendirmek maksadıyla, gerçek problemler üzerinde bazı deneyler yapılmıştır.

Anahtar Kelimeler: Tümevaran Mantıksal Programlama (TMP), Çok İlişkili Veri Madenciliği, Konsept Tanımlama, Kapsam, Doğruluk

To my dear wife, Sibel.

ACKNOWLEDGMENTS

This thesis is the product of the study spanning the past five and a half years. During this period, I have pleasant time and I am very happy to be a member of this team.

First of all, I would like to express my deepest gratitude to my supervisor Asst.Prof. Dr. Pınar Şenkul for all her guidance, encouragement, motivation and continuous support throughout my graduate studies. Without her great advice, help and understanding this work would never been possible.

I wish to express a lot of thanks to Prof. Dr. İsmail Hakkı Toroslu for his valuable suggestions and comments at crucial points in the development of the thesis and for finding the time in his very busy schedule to read our thesis.

Special thanks to Prof. Dr. Özgür Ulusoy for the cooperation and support he has provided throughout the steering meetings of this study.

I also want to express my gratefulness to my dear wife, Sibel, for all her patience, friendship and tolerance. Without her continuous support and encouragement, I would have never had the strength to complete this thesis. She looks after our sweet daughters Esra and Ayşegül very well.

I would like to thank to my parents Ali and Ayşe Kavurucu for their significant support throughout this work. Even though they have not directly influenced the thesis itself, they have provided me with apmle love, advice and support, both moral and financial not only during this work but all my life so far.

Finally, I am grateful to all my friends who have not influenced this thesis but have made my life happier during this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

[LIST OF ABBREVIATIONS]

ALEPH A Learning Engine for Proposing Hypothesis

ARM Association Rule Mining

CLAMF Classification with Aggregation of Multiple Features

C^2D Confidence-based Concept Discovery

CRIS Concept Rule Induction System

CWA Closed World Assumption

FOIL First Order Inductive Learner

GSG Generalised Selection Graph

ILP Inductive Logic Programming

KDD Knowledge Discovery in Databases

LGG Least General Generalization

MIS Model Inference System

MRDM Multi-Relational Data Mining

MRDTL Multi-Relational Decision Tree Learning

PTE Predictive Toxicology Evaluation

RLGG Relative Least General Generalization

RPT Relational Probability Tree

SG Selection Graph

SQL Structured Query Language

TILDE Top-down Induction of first-order Logical Decision Environment

CHAPTER 1

INTRODUCTION

Due to increase of complex data usage in information systems, the amount of data collected in relational databases is also increasing. This increase forced the development of multi-relational learning algorithms that can be applied to directly multi-relational data on the databases [26, 24]. Generally, first-order predicate logic is employed as the representation language for such learning systems. The learning systems, which find logical patterns valid for given background knowledge, have been investigated under a research area which is called Inductive Logic Programming (ILP) [68].

Concept discovery in relational databases is a predictive learning task. There is a specific target concept to be learned in the light of the past experiences. In ILP-based concept learning methods, logical patterns for the target concept are induced that are validated against the background facts. Association rule mining is a technique that is employed in the proposed algorithms for relational concept discovery. Association rule mining is finding frequent patterns, associations or correlations among sets of items or objects in databases. Relational association rules are expressed as query extensions in first-order logic [17, 20].

This thesis presents two new concept discovery methods, namely Confidence-based Concept Discovery (C^2D) and Concept Rule Induction System (CRIS), which are both predictive concept learning ILP systems that employ relational association rule mining concepts and techniques to find frequent and strong concept definitions according to given target relation and background knowledge. Both methods utilize absorption operator of 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 an APRIORI-based specialization operator based on confidence. C^2D constructs rules by examining the target concept instances in sequence, whereas CRIS considers the number of occurrences of constant arguments in the rule by taking all given target instances together into account. By this

way, the effect of target instance ordering on the concept discovery is eliminated and thus rule quality is further improved.

An important feature for a concept discovery method is the ability of incorporating aggregated information into the concept discovery process. In C^2D, well-known aggregate functions *COUNT, SUM, MIN, MAX* and *AVG* are defined in first-order logic and used as aggregate predicates for the situations where one-to-many relationships exist in the data set. Aggregation handling mechanism in CRIS considers the whole domain of an aggregated attribute, resulting in increase in the quality of the discovered rules in certain domains.

In these techniques, the traditional definition of confidence in Association Rule Mining (ARM) is modified to describe Closed World Assumption (CWA), a new pruning method based on the improved definition of confidence is defined in the APRIORI search lattice and a new hypothesis evaluation criteria is used to find rules that describe the target relation.

In this chapter, introductory information about the basic concepts related with the proposed algorithms is given.

1.1 Multi-Relational Data Mining

Advances in information technologies are making it possible to store increased volumes of data in digital form. The value of storing volumes of data depends on our ability to extract useful reports, find interesting events and trends, support decisions and policy based on statistical analysis and inference, and exploit the data to achieve business, operational, or scientific goals [30]. This gave arise to a research field called Knowledge Discovery in Databases (KDD). Since its beginning, the research made in this field has been vast and is continuously growing.

The terms KDD and Data Mining are sometimes used indistinctly. The process of KDD consists of three steps:

1. The first step is the pre-processing of the data set. The incorrect and missing data are removed and the format of the data set is converted into appropriate form for the data mining algorithm.

2. The second step is data mining, in which, the patterns and regularities in the data set are extracted.

3. In the third step, the results of the data mining process are translated into a more intel-

ligible format which is called post-processing.

KDD is essentially concerned with the nontrivial identification and extraction of valid, novel, potentially useful, and ultimately understandable knowledge from large databases. Data mining is the main step in this process [28].

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 are needed to store complex data for real life, data intensive applications. This has led to the need for multi-relational learning systems that directly apply to relational representations of structured problem domains. There are two key approaches in constructing relational learning systems: [26, 55]

1. In the first one, the method is composed of three parts: pre-processing, hypothesis construction and post-processing. In the pre-processing 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 into relational form. One of the ILP systems using this approach is the LINUS framework [56] that utilizes an embedded deductive hierarchical database (DHDB) in data transformation and one of the three propositional learning systems among ASSISTANT [11], NEWGEM [66] and CN2 [13] is used 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.

2. In the second one, attribute-value learning systems have been upgraded to the multi-relational counterparts in every branch of data-mining. ILP/RDM algorithms have many things in common with propositional learning algorithms. The difference between them is the representation of data, patterns and search techniques. 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 [71]. The propositional data structures used in data mining area, such as decision trees, if-then classification rules and association rules have been extended to relational form in multi-relational data mining (MRDM) systems [28, 24, 26]. Two most popular algorithms for inducing relational decision trees, SCART [50] and TILDE [9], are upgrades of the propositional decision tree induction systems, CART [51] and C4.5 [83], respectively. WARMR [18] upgrades the frequent item-set mining algorithm APRIORI [3] 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 [29] defines a relational distance measure, and then adapts k-nearest neighbour 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 [47]. The well known FOIL [82] system is an upgrade of the propositional rule induction program CN2 [14]. Another well known ILP system, PROGOL [69] can be viewed as upgrading the AQ approach [64] to rule induction.

Concept learning focused on developing search techniques that efficiently traverse target concept description space consisting of logical Horn clauses. There are various methods designed to solve this problem [5]:

- 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 making search using relative least general generalization (RLGG) operator.

FOIL [82] was the first relational learning algorithm that uses information gain based search heuristics. It uses an AQ-like covering approach [15] and it inherits the top-down

search strategy from MIS [92], which is an early concept learning system. Recently, many systems that extend FOIL in various aspects have been introduced such as FOCL [78].

CIA [15], MODELER [109] and RDT [46] 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. Relational patterns involve multiple relations from a relational database. They are typically expressed in subsets of first-order logic. A relation in a database corresponds to a predicate in first-order logic. The attributes of the predicate correspond to the arguments of the predicate.

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 [5].

MARVIN [88] 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. No search heuristics exist to direct the search; instead the oracle evaluates the quality of induced clauses.

CIGOL [72] employs three generalization operators based on inverse resolution, which are relational upgrades of absorption, intra-construction and truncation operators used in DUCE [74], 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 [69, 75] is a bottom-up Horn clause induction system, that uses the inverse entailment operator in induction phase. In the system, firstly the positive instance is selected to be generalized. And then, the most specific clause within the language constraints that entails the selected 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.

GOLEM [73] is a bottom-up ILP that is based on the relative least general generalization operator.

MRDM tools can be applied directly to multi-relational data to find relational patterns that involve multiple relations. However, most of the previous systems assume that the data reside in a single table and require preprocessing to integrate data in a single table before they can be applied. But, it can cause loss of meaning of information.

Just as many data mining algorithms come from the field of machine learning, many MRDM algorithms come from the field of Inductive Logic Programming (ILP) [68]. ILP has been concerned with finding patterns expressed as logic programs. In fact, a number of the ILP-based techniques proposed for MRDM rely on their propositional counterparts. Figure 1.1 [59] shows the relationship between propositional and MRDM algorithms proposed so far.

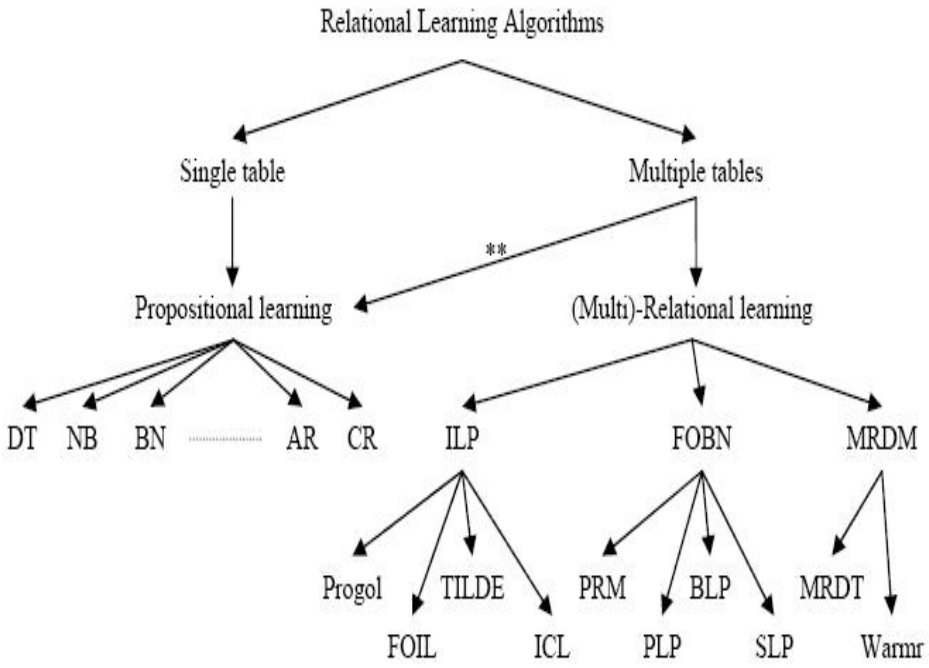

Figure 1.1: Relationships between propositional and multi-relational learning algorithms

1.2 ILP-based Learning

The amount of data collected on relational databases have been increasing due to increase in the use of complex data for real life applications. This motivated the development of multi-relational learning algorithms that can be applied to directly multi-relational data on the databases [26, 24]. For such learning systems, generally, first-order predicate logic is employed as the representation language. The learning systems, which induce logical patterns valid for given background knowledge, have been investigated under a research area, called Inductive Logic Programming (ILP) [68].

ILP systems can be classified into two general categories according to the learning technique: *predictive* 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 [34].

1.2.1 Predictive Inductive Learning

In *predictive* learning, there is a specific target concept to be learned in the light of past experiences. This is also a *concept discovery* task. Predictive learning can be applied to any classification or prediction problem, such as predicting carcinogenic activity of chemical compounds based on their chemical structures [97]. 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 learning task introduced by Muggleton in [67] 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 [55]. 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.

The aim of predictive 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 generally do not utilize the "support" concept in pruning the search space.

Predictive ILP systems learn the target concept via searching hypothesis space in one of two directions: top-down and bottom-up. Bottom-up approaches start with the most specific clause containing a given positive example and generalize the hypothesis until the concept description with the background knowledge implies all positive instances. On the other hand, top-down ILP systems begin with the most general hypothesis which covers all instances and noninstances of the concept and diminish the borders of the hypothesis such that the final hypothesis covers no negative instance of the concept. Besides, top-down (bottom-up) systems may employ a generalization (specialization) operator in order to adapt the hypothesis according to given concept instances [68].

1.2.2 Descriptive Inductive Learning

Descriptive data mining differs from the predictive data mining such that the search is not directed by a target concept. Descriptive ILP systems do not know which class or concept they are looking for in underlying database; instead they search 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 [84]. In other words, descriptive data mining systems explore 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).

The main aim in descriptive data mining 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 descriptive data mining systems.

Predictive ILP systems generally do not utilize the "support" concept in pruning the search space. However descriptive ILP systems like WARMR [18], which aims at identifying patterns without any specific target, utilize the general support rule, namely, APRIORI rule, as a strong search heuristics [19].

1.3 Motivation and Contributions

The motivation behind predictive ILP learning is to discover a complete and consistent hypothesis that best fits to the target concept instances. Each rule 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 rule. Predictive ILP systems generally do not utilize the "support" concept in pruning the search space; however descriptive systems APRIORI and WARMR utilize the general support rule, the APRIORI property, as a strong search heuristics. The need for the "support" concept in predictive ILP learning has led us to extend WARMR query mining tool into a rule mining system that discovers frequent and confident relational rules, including linearly recursive rules.

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 rules is also high since all the combinations of the added literal with the literals in the rule are needed to be frequent; therefore the system keeps track of infrequent rules 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 infre-

quent queries.

A challenging problem of relational concept discovery is dealing with intractably large search space. Several relational knowledge discovery systems have been developed employing various search strategies, heuristics, language pattern limitations and hypothesis evaluation criteria, in order to prune the search space. However, there is a trade-off between pruning the search space and generating high-quality patterns. Therefore, finding solutions to ease this trade-off is a basic motivation of this thesis.

In addition, another current drawback which also motivates research on a new system is the direct use of relational data. Most ILP-based 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.

In this thesis, two predictive concept learning ILP systems, namely Confidence-based Concept Discovery (C^2D) and Concept Rule Induction System (CRIS), are proposed which employ relational association rule mining concepts and techniques. They utilize absorption operator of inverse resolution for generalization of concept instances in the presence of background knowledge and refine these general patterns into frequent and strong concept definitions with an APRIORI-based specialization operator based on confidence.

C^2D and CRIS, first, generalize target concept instances in the presence of background knowledge as concept rules with one literal in the body (two literal rules) and populate first level of the search lattice with these generalizations so that each rule in the lattice covers at least one positive concept example. Absorption operator of inverse resolution introduced in [72], which is one of the most popular ILP generalization operators, is employed in this bottom-up induction step.

In the second step, the specialization operator joins two frequent queries that have all but one literal in common as in candidate generation step of APRIORI. By this way, the search lattice becomes more compact and, unlike WARMR, there is no need for keeping a list of infrequent queries. For search strategy and heuristics, C^2D and CRIS utilize breath-first search and relational version of the APRIORI rule, as in WARMR. But there is an extra bias in the proposed algorithms. In the specialization step, if the specialized rule has not higher confidence value than parent's values, then it is pruned such that it is not stronger than its parents in the correct path through the hypothesis. Because, in the correct path, each child rule must cover less negative examples (confidence value must be higher).

Another important property of C^2D and CRIS is the usage of aggregate predicates. The popular aggregate functions *COUNT, SUM, MIN, MAX* and *AVG* are defined in first-order logic and used as aggregate predicates if one-to-many relationships exist in the data set. Especially, the usage of the "COUNT", "MAX" and "MIN" aggregate functions are shown in the experiments.

The selection of target concept instance for generalization may effect the result hypothesis set. In order to prevent this, CRIS modifies the generalization step of C^2D to handle all the background facts in a more efficient way without distinguishing them as related or unrelated.

Major contributions of this thesis can be listed as follows:

1. The main difficulty in relational ILP systems is searching in intractably large hypothesis spaces. In order to cope with this problem, relational ILP systems put strong declarative biases on the semantics of hypotheses. In this thesis, we aimed to relax the declarative biases in such a way that body predicates may have variables which do not exist in the head predicate. On the other hand, in order to reduce the search space, a confidence-based pruning mechanism is used.

2. Many multi-relational rule induction systems require the user to determine the input-output modes of predicate arguments. Since mode declarations require a high level Prolog and domain knowledge, it is not meaningful to expect such a declaration from an ordinary user. Instead of this, we use the information about relationships between entities in the database if given. Therefore, in this thesis, the novel user knowledge about domain is not required.

3. Muggleton shows that [70], the expected error of an hypothesis according to positive versus all (positive and negative) examples do not have much difference if the number of examples is large enough. In other words, logic programs are learnable with arbitrarily low expected error from only positive examples. As relational databases contain only positive information, a pure multi-relational data mining system based on logic programming could be developed which relies on only positive instances stored as relations. Therefore, the proposed system directly works on relational database, without any requirement of negative instances.

4. The definition of confidence is modified to apply Closed World Assumption (CWA) in relational databases. We introduce type relations to the body of the rules in order to express CWA.

5. The choice of hypothesis evaluation criteria is an important factor on the quality of the generated patterns. In this thesis, we used an improved confidence-based hypothesis evaluation criterion, namely f-metric, which will be described in the following sections.

6. Previous ILP-based systems do not consider aggregate predicates in their concept description mechanisms. However, better rules (higher coverage and accuracy) can be discovered by using aggregate predicates in the background knowledge. To do this, aggregate predicates are defined in first-order logic and used in the proposed methods.

7. Numerical attributes are handled in a more efficient way. The rules having comparison operators on numerical attributes are defined and used in the main algorithm.

8. When the target concept has common attribute types with only some of the background predicates, the rest of the predicates (which are called *unrelated relations*) can never take part in hypothesis. This prevents the generation of transitive rules through such predicates. In order to solve this problem, the generalization mechanism of C^2D is extended in such a way that the indirectly related facts of the target concept instance are added to APRIORI lattice to allow transitive rules in the hypothesis. Moreover, CRIS prevents this drawback at the beginning efficiently.

9. The experiments show that the selection order of the target instance (the order in the target relation) may change the result hypothesis set in C^2D. In each coverage set, the induced rules depend on the selected target instance and the covered target instances in each step do not have any effect on the induced rules in the following coverage steps. To solve this problem, first, all possible values for each argument of a relation are determined by executing simple SQL statements in the database. Instead of selection a target instance, those values for each argument are used in the generalization step of CRIS.

1.4 Organization of the Thesis

This dissertation is divided into eight chapters.

- Chapter 1, the current one, discusses some reasons why multi-relational data mining and ILP-based learning become important. It also presents the motivation behind this thesis and the contributions of the proposed method.

- Chapter 2 gives preliminary information about ILP, Association Rule Mining (ARM) and aggregate functions. It describes the basic techniques in ILP which are used in the proposed method. In addition to this, the popular metrics used in ARM, which are support and confidence, and the popular aggregate functions are presented in this chapter.

- Chapter 3 presents a comparative overview of well-known ILP-based system's algorithms on a running example. These are LINUS, GOLEM, CIGOL, MIS, FOIL, PROGOL, ALEPH, WARMR and SAHILP. At the end of the chapter, these systems are compared with the proposed method.

- Chapter 4 explains the definition and usage of aggregate predicates in first logic on a well-known data set called Mutagenesis. Then, it presents the popular algorithm *Multi-Relational Decision Tree Learning (MRDTL)* and demonstrates the usage of aggregate predicates on it. Also, similar systems which supports aggregation are introduced.

- Chapter 5 gives the improved definition of confidence from relational database perspective and explains the proposed method using improved definition of confidence. In addition, it also presents aggregate predicate definition in C^2D and addition of indirectly related facts in the search space of the main algorithm for transitive rule construction.

- Chapter 6 introduces the improved version of C^2D (CRIS). It presents the differences between C^2D and CRIS algorithm by using the PTE-1 data set as an example.

- Chapter 7 discusses the experimental results of the proposed method on real world problems such as Learning Recursive Rules in Same-Generation Problem, Finite Element Mesh Design, Predictive Toxicology Evaluation, Mutagenicity Test, Finding Transitive Rules Using Unrelated Relations and Constructing Transitive Rules Under Missing Background Information.

- Chapter 8 includes concluding remarks and possible improvements.

CHAPTER 2

PRELIMINARIES

2.1 Basic Definitions

In this section, formal definitions of the basic concepts used throughout the dissertation are given.

2.1.1 Knowledge Representation Issues

We use first-order logic as the language to represent data and patterns. The formal definitions for data/pattern representation are given below:

Definition 2.1 *The smallest units in a theory are* variables *and* constants. *To distinguish* variable *names from* constant *names, we will use uppercase initial character for* variable *names and lowercase initial character for* constant *names.*

The following are variables: *A, B23*; and constants: *ab, yusuf.*

Definition 2.2 *A* term *can be either a constant, a variable, or an n-ary function symbol followed by a bracketed n-tuple of terms, where n ≥ 1 is called the arity of the* term.

The following are terms: *A, yusuf, f(A,B), g(C,D,b2).*

Definition 2.3 *A bracketed n-tuple of terms preceded by a predicate symbol is called an* atomic formula *or* atom.

The following is an atom *ancestor(A,yusuf)* with the predicate name *ancestor* and arity *2*.

Definition 2.4 *A* literal *is either an atom* $p(arg_1, ..., arg_n)$*, called a* positive literal, *or a negated atom* $\neg p(arg_1, ..., arg_n)$*, called a* negative literal.

The following are literals: *daughter(A,B), ¬ female(C).*

Definition 2.5 *Literals can be combined into logical formula by means of logical connectors* AND *(∧) to create a* conjunction*, and* OR *(∨) to create a* disjunction*.*

daughter(A,B) ∧ ¬ parent(B,A) ∨ female(A) is a formula.

Definition 2.6 *A* substitution θ *is a set* $\{X_1/t_1, ..., X_m/t_m\}$*, where each* X_i *is a variable such that* $X_i = X_j \Leftrightarrow i = j$*,* t_i *is a term different from* X_i*, and each element* X_i/t_i *is called a binding for variable* X_i*.*

Set {A/yusuf, B/ayse, C/D } is a substitution.

Definition 2.7 *A* quantification *of a variable A can either be universal, denoted by ∀A, or existential, denoted by ∃A.*

∀*A*∃*B*(daughter(A,B) ∧ parent(B,A)) is a formula, in which the variable *A* is universally quantified and the variable *B* is existentially quantified.

Definition 2.8 *A* clause *is a universally quantified disjunction* $\forall(l_1 \vee l_2 \vee ... \vee l_n)$*. When it is clear from the context that* clauses *are meant, the quantifier ∀ is dropped. A* clause $h_1 \vee h_2 \vee ... \vee h_p \vee b_1 \vee b_2 \vee ... \vee b_r$*, where the* h_i *are positive literals and the* b_j *are negative literals, can also be written as* $h_1 \vee h_2 \vee ... \vee h_p \leftarrow b_1 \wedge b_2 \wedge ... \wedge b_r$*, where* $h_1 \vee h_2 \vee ... \vee h_p$ *(*$p \geq 0$*) is called the* head *of the* clause*, and* $b_1 \wedge b_2 \wedge ... \wedge b_r$ *(*$r \geq 0$*) is called the* body *of the* clause*. This representation can be read as "*h_1 *or ... or* h_p *if* b_1 *and ... and* b_r*".*

∀(daughter(A,B) ∨ ¬ parent(B,A) ∨ female(A)) is a clause, also written as
daughter(A,B) ∨ female(A) ← parent(B,A)
where daughter(A,B) ∨ female(A) is the head, and parent(B,A) the body of the clause.

Definition 2.9 *A* definite clause *is a clause which only has one head literal. A definite clause with an empth body is called a* fact*. A* denial *is a clause with an empty head.*

daughter(A,B) ← female(A) ∧ parent(B,A) is a definite clause.

Definition 2.10 *A* query *is an existentially quantified conjunction* $\exists(l_1 \wedge l_2 \wedge ... \wedge l_n)$*. When it is clear from the context that* queries *are meant, the quantifier ∃ is dropped.*

$\exists$(daughter(A,B) $\wedge$ parent(B,A)) is a query.

A *query* $\exists(l_1 \wedge ... \wedge l_m)$ corresponds to the negation of a denial $\forall(\leftarrow l_1 \wedge ... \wedge l_m)$. In fact denials are often called *query clauses* or even queries.

Definition 2.11 *A* query extension *is an existentially quantified implification* $\exists(l_1 \wedge l_2 \wedge ... \wedge l_m) \rightarrow \exists(l_1 \wedge l_2 \wedge ... \wedge l_m \wedge l_{m+1} \wedge ... \wedge l_n)$*, with* $1 \leq m < n$*. To avoid confusion with clauses (which are also implications) we can write as* $l_1 \wedge l_2 \wedge ... \wedge l_m \rightsquigarrow l_{m+1} \wedge ... \wedge l_n$*. We call query* $l_1 \wedge l_2 \wedge ... \wedge l_m$ *the* body *and query* $l_{m+1} \wedge ... \wedge l_n$*) the* head *of the* query extension *[17]. In [20], relational association rules are called as* query extensions.

$\exists$(female(A) $\wedge$ parent(B,A)) $\rightsquigarrow$ $\exists$(daughter(A,B)) is a query extension.

Similar to queries, query extensions correspond to negated clauses. This can be described as:

$$l_1 \wedge ... \wedge l_m \rightsquigarrow l_{m+1} \wedge ... \wedge l_n$$

$$\Updownarrow$$

$$\neg(\exists(l_1 \wedge ... \wedge l_m) \wedge \forall(\neg l_{m+1} \vee ... \vee \neg l_n \leftarrow l_1 \wedge ... \wedge l_m))$$

Definition 2.12 *A* range-restricted query *is a query in which all variables that occur in negative literals also occur in at least one positive literal. A* range-restricted query extension *is a query extension such that both head and body are range-restricted queries.*

Definition 2.13 *A definite clause* C *θ-subsumes a definite clause* C′*, i.e. at least as general as* C′*, if and only if $\exists\theta$ such that:*

head(C) = *head*(C') *and body*$(C)\theta \subseteq$ *body*(C').

We use the above definition for concept rule generation which is described in Chapter 5.

Definition 2.14 *A* concept *is a set of frequent patterns, embedded in the features of the concept instances and relations of objects belong to the concept with other objects.*

In this thesis, we use the term *concept rule* (or shortly rule) to denote the association rule (range-restricted query extension), that is used for defining a concept [17, 20]. We also represent concept rules in definite clause format (h $\leftarrow$ b) throughout this dissertation, in which *h* is a single positive literal and *b* consists of positive literals.

2.1.2 Support and Confidence

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

Definition 2.15 *The* support *value of a concept rule* C *is defined as the number of different bindings for the variables in the head relation that satisfy the rule, divided by the number of different bindings for the variables in the head relation. In other words, it is the ratio of number of positive target instances captured by the rule over number of target instances.*

Let C be h ← b,

$$support(h \leftarrow b) = \frac{|bindings\ of\ variables\ for\ h\ that\ satisfy\ h \leftarrow b|}{|bindings\ of\ variables\ for\ h\ that\ satisfy\ h|} \tag{2.1}$$

Definition 2.16 *The* confidence *of a concept rule* C *is defined as the number of different bindings for the variables in the head relation that satisfy the rule, divided by the number of different bindings for the variables in the head relation that satisfy the body literals. In other words, it is the ratio of number of positive target instances captured by the rule over number of instances that are deducible by the body literals in the rule.*

Let C be h ← b,

$$confidence(h \leftarrow b) = \frac{|bindings\ of\ vars\ for\ h\ that\ satisfy\ h \leftarrow b|}{|bindings\ of\ vars\ for\ h\ that\ satisfy\ b|} \tag{2.2}$$

2.2 Inductive Logic Programming

2.2.1 Overview

Although logic has been studied for a long time, it was transformed into a mathematical science in the 19^{th} century. According to logical positivists, every mathematical statement can be phrased within the logical language of first-order predicate calculus and all valid scientific reasoning is based on logical derivation from a set of pre-conceived axioms. This is the basic idea behind deductive logic [68]. Those logical axioms, representing generalized beliefs, can be constructed from particular facts using inductive reasoning. Induction means reasoning

from specific to general. In the case of inductive learning from examples, the learner is given some examples from which general rules or a theory underlying the examples are derived.

Stephen Muggleton introduced Inductive Logic Programming (ILP) that is the intersection of machine learning and logic programming [68]. ILP studies learning from examples, within the framework provided by clausal logic. In ILP systems, the training examples, the background knowledge and the induced hypothesis are all expressed in a logic program form. There are two types of examples: *positive (true)* and *negative (false)*. They are usually given as ground atoms but sometimes ground clauses can be used as examples, as well. Both background knowledge and the induced theory are represented as finite sets of clauses.

Definition 2.17 *A hypothesis is* complete *with respect to background knowledge and training examples if all the positive examples are covered.*

Definition 2.18 *A hypothesis is* consistent *with respect to background knowledge and training examples if none of the negative examples are covered.*

Two measures are used to test the quality of the induced theory. After learning, the theory with background knowledge should be complete and consistent. Completeness and consistency together form correctness. At the end, it induces concepts or frequent patterns as logical expressions. The term *hypothesis* is also used for induced concept/pattern description.

Definition 2.19 Inductive Concept Learning *is the task of learning a hypothesis from a set of training examples and background knowledge, such that the induced hypothesis is complete and consistent.*

Inductive Concept Learning is in fact searching for complete and consistent concept descriptions in the space limited by description language of the ILP system [65]. The current state of art in ILP aims 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 [105].

The most commonly addressed task in ILP is the task of learning logical definitions of relations, where tuples that belong or do not belong to the target relation are given as examples [27]. From training examples ILP then induces a logic program (predicate definition) corresponding to a view that defines the target relation in terms of other relations that are given as background knowledge.

2.2.2 Basic ILP Techniques

2.2.2.1 Search Strategies

Two basic steps in the search for a correct theory are *specialization* and *generalization* [55].

Definition 2.20 *If a theory covers negative examples, it means that it is too strong, it needs to be weakened. In other words, a more specific theory should be generated. This process is called* specialization.

Definition 2.21 *On the other hand, if a theory does not imply all positive examples, it means that it is too weak, it needs to be strengthened. In other words, a more general theory should be generated. This process is called* generalization.

Specialization and generalization steps are repeated to adjust the induced theory in the overall learning process.

There are two approaches for the search direction: *top-down* and *bottom-up*.

Definition 2.22 Top-down *approach starts with an overly general theory and tries to specialize it until it no longer covers negative examples.*

Specialization (refinement) operators employ two basic operations on a clause: apply a substitution to the clause and add a literal to the body of the clause. *Refinement graph* is the most popular data structure used in specialization. An analysis of refinement operators in ILP is described in [106].

Definition 2.23 Bottom-up *approach starts with an overly specific theory and tries to generalize it until it can not further be generalized without covering negative examples.*

Generalization operators perform two basic syntactic operations on a clause: apply an inverse substitution to the clause and remove a literal from the body of the clause. *Relative least general generalization (rlgg)* (used in GOLEM) and *inverse resolution* (used in CIGOL) are two basic generalization techniques.

2.2.2.2 Generalization Techniques

Definition 2.24 *Clause C θ-subsumes clause C' if there is a substitution θ that can be applied to C such that every literal in $C\theta$ occurs in C'. If $\theta = \phi$ and $C \leq C'$ (C is at least as general as C'), then C is a subset of C'; otherwise if $\theta \neq \phi$ and $C \leq C'$, then C is a subset of $C'\theta^{-1}$ ($(C\theta \subseteq C') \equiv (C \subseteq C'\theta^{-1})$).*

Generalization operators under θ-subsumption perform two syntactic generalization operations: obtain C by applying inverse substitution to the clause C′ and/or by removing one or more literals from the clause C′.

There are two basic subsumption based generalization operators: relative least general generalization developed by Plotkin [81] (used in GOLEM) and inverse resolution introduced by Muggleton and Buntine [72] (used in CIGOL).

Definition 2.25 *The* least general generalization (lgg) *of two clauses C_1 and C_2, denoted by* lgg(C_1, C_2)*, is the least upper bound of C_1 and C_2 in the θ-subsumption lattice.*

To actually compute the lgg of two clauses, lgg of terms, atoms and literals need to be defined first [55]:

Definition 2.26 *Lgg of terms lgg(t_1, t_2):*

1. lgg(t, t) = t,

2. lgg(s, t) = V, where $s \neq t$ and at least one of s and t is a variable; in this case, V is a variable which represents lgg(s, t),

3. lgg($f(s_1, ..., s_n)$, $f(t_1, ..., t_n)$) = $f(lgg(s_1, t_1), ..., lgg(s_n, t_n))$,

4. lgg($f(s_1, ..., s_n)$, $f(t_1, ..., t_n)$) = V, where $f \neq g$.

For example, lgg([a, b, c], [a, c, d]) = [a, X, Y].

Definition 2.27 *Lgg of atoms lgg(A_1, A_2):*

1. lgg($p(s_1, ..., s_n)$, $p(t_1, ..., t_n)$ = $p(lgg(s_1, t_1), ..., lgg(s_n, t_n))$,

2. lgg($p(s_1, ..., s_n)$, $q(t_1, ..., t_n)$ is undefined if $p \neq q$.

Definition 2.28 *Lgg of literals lgg(L_1, L_2):*

1. if L_1 and L_2 are atoms, then lgg(L_1, L_2) is computed as defined above,

2. if both L_1 and L_2 are negative literals, $L_1 = \neg A_1$ and $L_2 = \neg A_2$, then lgg(L_1, $L_2 = \neg$ lgg(A_1, A_2),

3. if L_1 is a positive and L_2 is a negative literal, or vice versa, lgg(L_1, L_1) is undefined.

Definition 2.29 *Lgg of clauses lgg(C_1, C_2):*

Let $C_1 = L_1, ..., L_n$ and $C_2 = K_1, ..., K_m$. Then, lgg(C_1, C_2) = L_i j = lgg(L_i, K_j) — $L_i \in C_1$, $K_j \in C_2$ and lgg(L_i, K_j) is defined.

Given the positive examples e_1 and e_2 and the background knowledge B, the lgg of e_1 and e_2 with respect to B is computed as:

Definition 2.30 *rlgg(e_1, e_2) = lgg(($e_1 \leftarrow K$), ($e_2 \leftarrow K$)),*

where K denotes the conjunction of the background facts.

Inverse resolution is built on the fact that induction is the reverse operation of deduction [72]. The resolution rule of deductive inference allows to derive a resolvent clause C entailed from two given parent clauses C_1 and C_2, such that C_1 contains the literal L_1 and C_2 contains L_2 and $\theta = \theta_1\theta_2$ is the most general unifier of $\neg L_1$ and L_2, employing SLD-resolution procedure [62] as follows:

$$C = (C_1 - L_1)\theta_1 \cup (C_2 - L_2)\theta_2 \tag{2.3}$$

The inverse resolution inverts the resolution process by generalizing C_1 from C and C_2.

Muggleton and Buntine employed three types of generalization operators based on inverse resolution in CIGOL system: $\vee$-*operator* (*absorption operator*), W-operator and the truncation operator. The proposed method, C^2D, utilizes the $\vee$-operator in generalizing concept instances using background knowledge.

Definition 2.31 *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 $\{C_1, C_2\}$ [72].*

In contrast to the resolution, the $\vee$-operator derives one of the clauses on one arm of the $\vee$ tree, C_2, given the clause on the other arm, C_1, and the base clause C. 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 [72]. 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 2.3 as follows:

$$C_2 = (C \cup \neg(C_1 - L_1)\theta_1)\theta_2^{-1} \cup L_2 \tag{2.4}$$

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:

$$L_2 = \neg L_1\theta_1\theta_2^{-1} \tag{2.5}$$

Substituting Equation 2.5 into Equation 2.4

$$C_2 = (C \cup \neg(C_1 - L_1)\theta_1)\theta_2^{-1} \cup \neg L_1\theta_1\theta_2^{-1} \qquad = (C \cup \neg C_1\theta_1)\theta_2^{-1} \tag{2.6}$$

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 2.6. If the background knowledge C_1 is represented by ground unit clauses, i.e. $C_1 = L_1$ and $\theta_1 = \phi$ then Equation 2.6 becomes:

$$C_2 = (C \cup \neg L_1)\theta_2^{-1} = (C \cup \neg C_1)\theta_2^{-1} \tag{2.7}$$

Therefore, the indeterminacy is reduced to the choice of the inverse substitution θ_2^{-1}. 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.

2.2.2.3 Specialization Techniques

Specialization techniques search the hypothesis space in a top-down manner. The basic specialization ILP technique is top-down search of refinement graphs.

Definition 2.32 *A* refinement graph *is a directed, acyclic graph in which nodes are program clauses and arcs correspond to the basic refinement operations: substituting a variable with a term, and adding a literal to the body of the clause.*

Search of the refinement graph starts with the most general clause and continues by searching clause refinements in a breath-first manner. At each step, all minimal refinements are generated and tested for coverage. The acceptable refinements must cover the selected positive example. The process stops when the first acceptable consistent clause is found [55].

In this thesis, we use the refinement graph structure as a graph consists of query extensions.

2.2.2.4 Pruning Techniques

Definition 2.33 *Any mechanism employed by a learning system to constrain the search for hypothesis is called as* bias *[105].*

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. Almost all ILP systems use *language bias* [110].

After the borders of the search space are determined by language bias, the *search bias* restricts 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 [61]. The efficiency considerations can not tolerate this exhaustive search; therefore, in most of the systems, some filtering methods to prune the space are utilized.

In some interactive systems, an oracle determines the soundness criteria of induced rules in the learning phase explicitly or implicitly [55]. 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 hypothesis invented through search steps.

In general, there is a trade-off between the efficiency of an ILP system and the quality of the theory, and the degree of efficiency vs. quality is defined with the use of biases.

2.3 Association Rules

The data mining systems aim at extracting knowledge about the huge amount of stored data in favor of data owner for improving business gain. 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 [2, 4, 37, 89].

The relations between the attributes of a single table are represented by *Boolean association rules*. *Boolean association rules* are in fact propositional classification rules with no single target attribute, in the form of A $\Rightarrow$ B where A and B are a set of conditions with no restriction on the consequent B [34]. The relations between the attributes of a single table are

represented by *boolean association rules*.

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 [103].

Definition 2.34 *A* frequent item set *is defined as an item set with support value greater than a support threshold.*

Definition 2.35 Support *of an item set is the frequency of the item set, defined in terms of the fraction of the baskets including item set.* Support *of an association rule $A \Rightarrow B$ is the* support *of the item set $A \cup B$.*

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.

Definition 2.36 Confidence *of an association rule $A \Rightarrow B$ is evaluated by the probability of the baskets, having the item set A, also have items B. In other words, the* confidence *of an association rule is the ratio of the number of baskets that contain the item set $A \cup B$ to the number of baskets including the item set A.*

Definition 2.37 *A* strong association rule *is a rule that has confidence greater than a confidence threshold.*

Support shows the generality of the rule and the *confidence* designates the validity of the rule.

The most popular and well known association rule mining algorithm, as introduced in [4], is *APRIORI*. APRIORI utilizes an important property of frequent item sets in order to prune candidate item set space:

Property 2.1 *All subsets of a frequent item set must be frequent.*

The contra-positive of this property says that if an item set is not frequent than 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

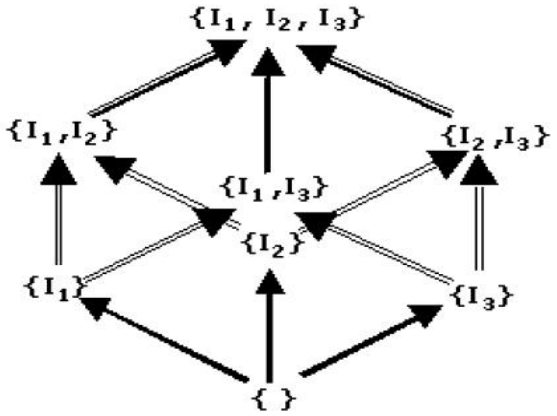

Figure 2.1: The APRIORI lattice with three items

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 2.1 [100].

The item set lattice in Figure 2.1 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 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.

Definition 2.38 *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 [100]:

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 min_sup (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 $L_1 \cup L_2 \cup ... \cup L_n$.

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.

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 potentially recurrent relational patterns in a relational database. In *relational association rule mining* systems, generally, relational upgrade of APRIORI rule is employed, as the search heuristics.

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.

Definition 2.39 *The* support *of a relational association rule, $A \sim> B$, is the* support *of the predicate set $A \cup B$ in the rule. The* confidence *of a relational association rule is defined as the* support *of the pattern $A \cup B$ divided by the* support *of the body A.*

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. However, in this thesis, we always choose attributes of the head predicate as key objects, we will not discuss this issue further.

2.4 Aggregate Functions

Aggregate functions provide a rich mechanism for expressing the characteristics of the relations having one-to-many relationships among them. Such relationships are common in databases. In concept discovery, conditions on aggregation such as *count* < 10 or *sum* > 100 may define the basic characteristic of a given concept better. For this reason, in this thesis, we extend the background knowledge with aggregate predicates in order to characterize the structural information that is stored in tables and associations between them.

Aggregate functions takes as input as a set of records in a database, and produces a single value as output [48]. As an example, it may be important to calculate *average* result for each

class after a midterm exam. To calculate these values, the following generic SQL statement can be used:

```
SELECT class-id, AVG(result)
FROM midterm-results
GROUP BY class-id
```

Alternatively, aggregate functions can be used to project information stored in several tables on one of these tables, essentially adding virtual attributes to this table. In the case where the information is projected on the target table, and structural information belonging to an individual is summarized as a new feature of that individual, aggregate functions can be thought of as a form of feature construction. For example, aggregate functions can be defined on one-to-many relationships between two tables. Assume, the table *t1* has a primary key *arg1* and the table *t2* has an argument *arg2* which is a foreign key according to the argument *arg1* of *t1*. Then, an aggregate function can be applied to *t2* which will return a value for each different value of argument *arg1* of *t1*. These values can be defined as a virtual attribute of the table *t1*. To calculate these values, the following generic SQL statement can be used:

```
SELECT t2.arg2, SUM(t2.arg3)
FROM t1, t2
WHERE t1.arg1 = t2.arg2
GROUP BY t2.arg2
```

The *COUNT*, *AVG*, *SUM*, *MIN* and *MAX* are the popular aggregate functions used in data mining. An overview of interesting classes of aggregate functions, along with a hierarchy of increasing complexity is given in [79].

CHAPTER 3

A COMPERATIVE STUDY ON CONCEPT DISCOVERY SYSTEMS

ILP techniques are widely used for classification and concept discovery in the data mining algorithms. In classification, general rules are created according to data and then they are used for grouping the unclassified data. In concept discovery, interesting rules describing the concept, if exist, are given to the users of the system. Several ILP-based systems are developed which employs various search strategies, heuristics and language pattern limitations. LINUS, GOLEM, CIGOL, MIS, FOIL, PROGOL, ALEPH, WARMR and SAHILP are well-known concept discovery systems which employ ILP techniques in their process.

In this section, the above-mentioned systems are described and the fundamentals of their concept discovery mechanisms are demonstrated on a running example.

There are also some other systems given in [12, 53, 104, 102], which use logic in their algorithm, but they are not described in detail in this chapter.

3.1 Example Data Set

The *daughter* relation given in Table 3.1, is used as a running example throughout this thesis. In this example, *daughter (d)* is the concept to be learned, and four concept instances are given. Background facts of two relations, namely *parent (p)* and *female (f)* are provided. Finally, types of the attributes of relations are listed.

In the example data set, there are two positive and two negative concept instances. From relational database perspective, the negative concept instances do not exist in the *daughter* table (the proposed techniques do not process negative data). The task is to define target relation *daughter(A, B)*, which states that person A is a *daughter* of person B, in terms of the

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

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

background knowledge relations *parent* and *female*. The type relation *person* is defined as:

person = {*ann, mary, tom, eve*}.

3.2 Bottom-Up ILP Systems

3.2.1 GOLEM

GOLEM [73] is a bottom-up relational ILP system, which constraints the search space by using *rlgg* operator that is based on Plotkin's notion of relative least general generalization [81] (rlgg is described in Section 2.2.2).

The definition of rlgg is as follows:

$$rlgg(e_1, e_2) = lgg((e_1 \leftarrow K), (e_2 \leftarrow K)). \tag{3.1}$$

where e_1 and e_2 are two positive examples, and K is the conjunction of the background facts. Since, such a clause can contain infinitely many literals, it uses some constraints when introducing new variables in the body of the *rlgg*. The variables in the body of the *rlgg* have to be, directly or indirectly, uniquely determined by the values of the variables in the head. In addition to this, it uses negative examples and mode declarations to reduce the size of the clauses.

The *rlgg* of e_1=daughter(mary, ann) and e_2=daughter(eve, tom) from the example data set is computed as follows (Note that constants are abbreviated as a=ann, m=mary, e=eve, t=tom):

K = parent(a, m), parent(a,t), parent(t,e), female(a), female(m), female(e).

C_1 = daughter(A,B) ← parent(B,A), parent(B,C), parent(C,D), female(B), female(A), female(D).

C_2 = daughter(A,B) ← parent(C,D), parent(C,B), parent(B,A), female(C), female(D), female(A).

The result is lgg of C_1 and C_2:

daughter(A, B) ← parent(B, A), female(A), female(D), parent(C, D).

Since the variables of female(D) and parent(C, D) do not directly or indirectly determined by the values of the variables in the head, they are irrelevant and the final result is:

daughter(A, B) ← parent(B, A), female(A).

In order to generate a single clause, GOLEM first randomly picks several pairs of positive examples, computes their rlggs and chooses the one with greatest coverage. If the final clause does not cover all positives, the covering approach will be applied. The covered positives are removed from the input and the algorithm will be applied to the remaining positives [73, 55].

3.2.2 CIGOL

CIGOL (logic backwards) [72] is a bottom-up relational ILP system, which is based on inverse resolution. The basic idea is to invert the resolution rule of deductive inference using the generalization operator based on inverse substitution.

Four operators in inverse resolution are absorption, identification, intra-construction and inter-construction. CIGOL uses the absorption operator. The absorption operator is defined as follows:

$$Infer\text{“}p \leftarrow q, B\text{”} from\text{“}p \leftarrow A, B \wedge q \leftarrow A\text{”}. \tag{3.2}$$

For the example data set, at the beginning, the hypothesis H is equal to empty set (H=ϕ). CIGOL encounters the first positive example “daughter(mary, ann)” (e_1). It seeks for a clause C_1 which entails e_1 together with a background fact. Related background facts are {parent(ann, mary), parent(ann, tom), female(ann), female(mary)}. The acceptable C_1 clauses are as follows:

daughter(mary, B) ← parent(B, mary).

daughter(mary, B) ← parent(B, tom).

daughter(mary, B) ← female(B).

daughter(A, ann) ← parent(ann, A).

daughter(A, ann) ← female(A).

Then it constructs another clause C_2, by applying inverse resolution to generated C_1 clauses. For example, for C_1,

C_1 = daughter(mary, Y) ← parent(Y, mary),

CIGOL tries all possible inverse resolutions on C_1 by checking background facts having *mary*. By taking female(mary) into account as a related fact, applying inverse substition θ^{-1} = {mary/A}, it finds the following clause C_2, which covers all the positive examples:

C_2 = daughter(A, B) ← parent(B, A), female(A).

3.3 Top-Down ILP Systems

3.3.1 LINUS

Early ILP systems were mainly attribute-value learners, in which propositional logic is used. LINUS [56] was one of the most popular systems in this category. It is an ILP system, integrating several ILP attribute-value learning algorithms in a single environment. It can be viewed as a toolkit, in which one or more of the algorithms can be selected in order to find the best solution for the input. It is non-incremental [1] and non-interactive [2].

The main algorithm behind LINUS consists of three steps:

1. In the first step, the learning problem is transformed from relational to attribute-value form.

2. In the second step, the transformed learning problem is solved by an attribute-value learning method.

3. In the final step, the induced hypothesis is transformed back into relational form.

For the example data set, in the first step, possible applications of the background predicates on the arguments of the target relation are determined, taking the argument types into account. Each such application introduces a new attribute. The corresponding attribute-value learning problem is shown in Table 3.2.

[1] In incremental learning, the examples are given one by one and the system adjusts its theory each time

[2] Interactive systems can interact the user in order to obtain additional information

Table 3.2: The attribute-value transformation of the example data set

C	Variables		Propositional Features					
	A	**B**	**p(A, A)**	**p(A, B)**	**p(B, A)**	**p(B, B)**	**f(A)**	**f(B)**
+	mary	ann	F	F	T	F	T	T
+	eve	tom	F	F	T	F	T	F
-	tom	ann	F	F	T	F	F	T
-	eve	ann	F	F	F	F	T	T

In the second step, a decision tree learning algorithm will be applied to the transformed form of the example data set. In this step, the following if-then-rule will be induced:

If $(parent(B, A) = T)) \wedge (female(A) = T)$ then $daughter(A, B) = T$.

In the final step, the induced rule will be transformed into relational form as follows:

daughter(A, B) $\leftarrow$ parent(B, A), female(A).

3.3.2 MIS

Model Inference System (MIS) is a top-down relational ILP system, which uses refinement graph in the search process [92].

MIS is an interactive and incremental system that can accept new training examples. In its algorithm, initially, the hypothesis set is empty (H=ϕ). Then it reads the examples (either positive or negative) one by one. If the example is negative and covered by some clauses in the hypothesis set, then incorrect clauses are removed from the solution set. If the example is positive and is not covered by any clause in the solution set, with breadth-first search, a clause *c*, which covers the example, is developed and added to solution set. The process will continue until the solution set (*H*) becomes complete and consistent [55].

For the example data set, the search stars with the most general clause "C_1=daughter(A, B) $\leftarrow$ ". It covers the first positive example daughter(mary ann) and is put into *H*. The second example daughter(eve, tom) is positive and covered by C_1. The third example daughter(tom, ann) is negative and covered by C_1. Therefore, C_1 is removed from *H*. Then, the refinements of C_1 are generated as "daughter(A, B) $\leftarrow$ L ". The literals in *L* can be {A=B, female(A), female(B), female(C), parent(A,A), parent(A,B), parent(B,A), parent(B,B), parent(A,C), parent(C,A), parent(B,C), parent(C,B)}. Then, these refinements are checked one by one for completeness and consistency. As none of the refinements are complete and consistent, the

refinements of the literals are generated and tested. Part of the refinement graph for the example data set is shown in Figure 3.1.

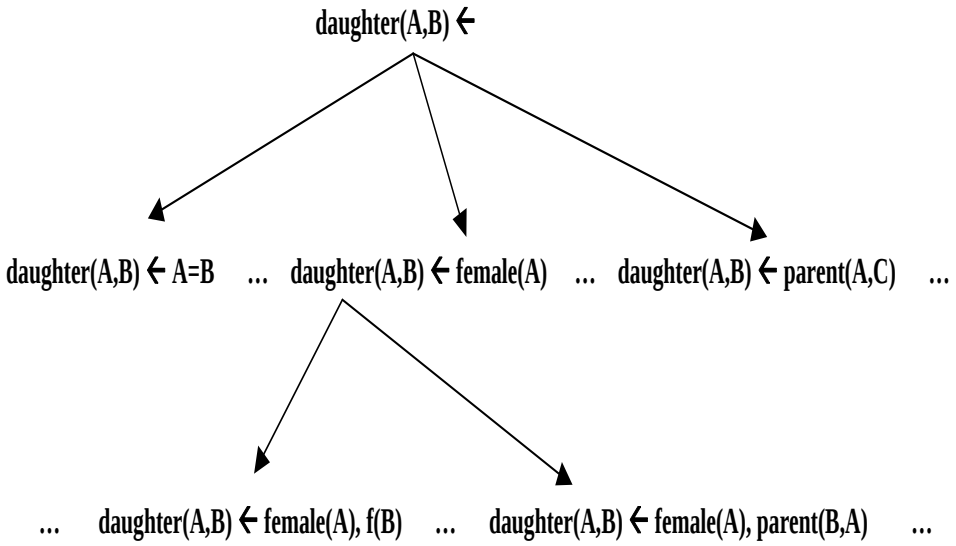

Figure 3.1: Part of an refinement graph

At the end, it finds the clause;

daughter(A, B) ← parent(B, A), female(A).

which is complete and consistent. If a consistent but not complete clause is found, the positives covered are removed and the algorithm starts seeking for solution for the remaining positive examples.

3.3.3 FOIL

First-Order Inductive Learner (FOIL) is a top-down relational ILP system, which uses refinement graph in the search process as in MIS. It uses the covering approach for solutions having more than one clause. It is a non-incremental and non-interactive system [82].

It allows recursive clauses in the solution. In addition to this, negative examples are not necessarily given to FOIL. It can generate them based on the CWA at the beginning. At the end, irrelevant clauses are removed from the solution set. Negative literals are allowed in the clauses of the solution set.

It starts with an empty body and adds a literal at each step according to the information gain. It examines all possible refinements of the current clause and checks the values of weighted information gain value. "$I(C) = -log_2 \frac{n(+)}{n}$" is the information gain value of the clause *C* where *n(+)* is the number of positive examples covered by *C* and *n* is the number of all examples covered by *C*. If literal *L* is added to *C* and a new clause C_2 is produced, then $I(C_2)$ is calculated. Let *m* be the number of positive examples both covered by *C* and C_2. Gain of adding *L* to *C* is equal to:

$$Gain(L) = m \times (I(C)-I(C_2))$$

The literal *L* having the maximum gain value is selected in the specialization process. If the new clause is consistent, then the positives covered by that clause are removed and the search process starts from the beginning for the remaining positive examples. Otherwise, specialization process continues until the clause becomes consistent (*I(C)=0*).

For the example data set, the search stars with the most general clause "C_1=daughter(A, B) ← ". The number of positives covered is 2 (n(+)=2) and the number of negatives covered is 2 (n(-)=2). The sample clauses that can be generated from C_1 and their gain values are given in Table 3.3.

Table 3.3: Sample clauses generated in FOIL

Name	**Clause**	**I(C_i)**	**Gain of Literal L**
C_1	daughter(A, B) ←.	1.0	
C_2	daughter(A, B) ← A=B.	0	Gain(A=B)=0
C_3	daughter(A, B) ← female(A).	0.58	Gain(female(A))=0.84
C_4	daughter(A, B) ← female(B).	0.3	Gain(female(B))=0.7
	...		
C_8	... daughter(A, B) ← parent(B,A).	0.58	Gain(parent(B,A))=0.84
	...		

The clauses having literals female(A) and parent(B, A) have maximum gain. Therefore, their refinements are checked for the maximum gain. At the end, the clause

C_n=daughter(A, B) ← parent(B, A), female(A).

with information gain as 0 is found which is complete and consistent. Therefore, the search process stops.

3.3.4 PROGOL

PROGOL [69, 75] is a top-down relational ILP system, which is based on inverse entailment. It performs a search through the refinement graph. In order to prune the hypothesis space, besides a definite program B as background knowledge and a set of ground facts E as examples, PROGOL requires a set of mode declarations as input. It creates the most specific clause (also called bottom clause) as the greatest lower bound of the refinement graph during pruning.

A bottom clause is a maximally specific clause ($\perp$), which entails (covers) a positive example *e* and is derived using inverse entailment. PROGOL starts the search with empty body, and goes through in the refinement lattice, which has literals that are elements of the bottom clause. PROGOL chooses the clause having maximum f value. The definition of f value is given below:

Definition 3.1 *f = p - (n + c + h), where p is the number of positives deducible from the clause, n is the number of negatives deducible from the clause, c is one less than the length of clause and h is the number of further atoms to complete the clause.*

The mode declarations define the predicates from B, which can appear in the head and in the body of the clauses in the hypothesis space, as well as the type of the arguments that are valid for the predicates. A mode declaration has either the form *modeh(n, atom)* or *modeb(n, atom)* where *n*, the recall, is either an integer, $n \geq 1$, or '*' and *atom* is a ground atom. The recall is used to bound the number of alternative solutions for instantiating the *atom* (default is '*' representing all solutions). Terms in the *atom* are either *normal* or *place-marker*. A *normal term* is either a constant or a function symbol followed by a bracketed tuple of terms. A *place-marker* is either *+type, -type or #type*, where *type* is a constant (+ is used for input variables, - for output variables and # for constant values). If *m* is a mode declaration then *a(m)* denotes the *atom* of *m* with place-markers replaced by distinct variables. The sign of *m* is positive if *m* is a *modeh* and negative if *m* is *modeb*.

For the example data set, assume that the mode declarations are as given in Table 3.4.

For the example data set, the first positive example is daughter(mary, ann). In inverse entailment;

Table 3.4: Mode declarations for the daughter data set

Mode Declaration	**Explanation**
modeh(1,daughter(+person, +person))	daughter is head relation, for two input arguments, daughter returns one result, either true or false
modeb(*,parent(+person, -person))	parent is a body relation, for the first input argument, there can be more than one child
modeb(*,parent(-person, +person))	parent is a body relation, for the second input argument, there can be more than one parent
modeb(1,female(+person))	female is a body relation, for the input argument, female returns one result, either true or false
modeb(1,female(-person))	female is a body relation, for the output argument, female returns one result, either true or false

$B \wedge H \models E$

$B \wedge \neg E \models \neg H$

$B \wedge \neg E \models \neg \perp \models \neg H$

$H \models \perp$.

From the head mode declaration we have the trivial deduction:

$B \wedge \neg E \models \neg$daughter(mary, ann)

From the body mode declarations we have the following deductions:

$B \wedge \neg E \models \neg$parent(ann,mary)

$B \wedge \neg E \models \neg$female(ann)

$B \wedge \neg E \models \neg$female(mary)

$B \wedge \neg E \models \neg$parent(ann,tom)

$B \wedge \neg E \models \neg$female(eve)

$B \wedge \neg E \models \neg$parent(tom,eve)

The bottom clause is:

daughter(A, B) ← parent(B, A), female(B), female(A), parent(B, C),
female(C), parent(D, C).

The solution must consist of the sub-elements of the bottom clause. The refinements of the initial clause daughter(A, B) ← . are:

daughter(A, B) ← . (p=2, n=2, c=0, h=2, f=-2)
daughter(A, B) ← parent(B, A). (p=2, n=1, c=1, h=0, f=0)
daughter(A, B) ← female(B). (p=1, n=2, c=1, h=1, f=-3)
daughter(A, B) ← female(A). (p=2, n=1, c=1, h=1, f=-1)
daughter(A, B) ← parent(B, C). (p=2, n=2, c=1, h=2, f=-3)
daughter(A, B) ← female(C). (p=2, n=2, c=1, h=3, f=-4)
daughter(A, B) ← parent(D, C). (p=2, n=2, c=1, h=4, f=-5)

It chooses daughter(A, B) ← parent(B, A), but it has non-negative n value. Therefore, it further refines it by adding literals from the bottom clause. Finally it finds the clause

C_n=daughter(A, B) ← parent(B, A), female(A).

with f=0 and n=0. As it covers all the positives, the search stops. If it does not cover all the positives, as the covering approach supposes, deducible positives are removed from set E and the process starts from the beginning for the remaining examples.

3.3.5 ALEPH

A Learning Engine for Proposing Hypotheses (ALEPH) [93] is a top-down relational ILP system based on inverse entailment similar to PROGOL. The basic algorithm is the same as PROGOL algorithm whereas it is possible to apply different search strategies, evaluation functions and refinement operators. It is also possible to define more settings in ALEPH such as minimum confidence and support.

Minpos and *minacc* are the two parameters representing minimum support and confidence criteria in ALEPH. The default value for *minpos* is 1 and it sets a lower bound on the number of positive examples to be covered by an acceptable clause. If the best clause covers positive examples below this number, then it is not added to the current theory. The default value for *minacc* is 0.0 (possible values are floating-point numbers between 0 and 1) and it sets a lower bound on the minimum accuracy of an acceptable clause.

For the example data set, execution and result of the algorithm is the same as in Progol.

3.3.6 WARMR

Design of algorithms for frequent pattern discovery has become a popular topic in data mining. Almost all algorithms have the same of level-wise search known as APRIORI algorithm [3]. The level-wise algorithm is based on a breadth-first search in the lattice spanned by a specialization relation between patterns [18, 20].

The APRIORI method searches one level of the lattice at a time. It starts from the most general pattern. It iterates between candidate generation and candidate evaluation phases. In candidate generation, the lattice is used for pruning non-frequent patterns from the next level. In candidate evaluation, frequencies of candidates are computed with respect to the database. Pruning is based on the monotonicity property with respect to frequency: if a pattern is not frequent then none of its specializations are frequent.

WARMR [18, 20] is a descriptive ILP system that employs APRIORI rule as search heuristics. Therefore, it is not a predictive system, i.e. it does not define the target relation. Instead, it can find the frequent queries including the target relation. Then, it is possible to extract association rules having target relation in the head according to confidence criteria. The target relation is defined as the key relation in WARMR.

In WARMR algorithm, at the beginning there are three sets: candidate queries (Q), frequent queries (F) and infrequent queries (I). Q is initialized as having the key predicate. F and I are initialized as empty set. In the first level, the specializations of the item in Q are generated according to language bias (warmode is similar to mode declaration in PROGOL). They are put into current Q set. After this, frequency values of the items in Q are evaluated and infrequent items are put into I and frequent items are put into F. In the next level, Q set is generated according to previous contents of Q, F and I set.

The language bias (warmode) defines the types and modes of the parameters of the predicates. The user can define the warmode in the settings file in the input data. Input-output modes of the variables in the formalism constrain the refinement of queries in a way that the modes the determine which atoms can be added to a query [26]. The key predicate of frequent patterns is specified in the formalism, too. Additionally, the types of the variables can be declared as in PROGOL [69]. An example declarative language bias specification in warmode notation for the example data set is illustrated below.

Key = daughter(-) Atoms = parent(+,-), parent(-,+), female(-), female(+)

As an analogy to to the APRIORI method, 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 3.5 [20], thus employs a level-wise search such that specific candidate queries Q_2 are generated from simpler, general frequent queries Q_1 where Q_1 θ-subsumes Q_2 [100].

Table 3.5: The pseudo code of the WARMR algorithm

Inputs: database r; Warmode language L; support threshold min_sup
Outputs: all queries $Q \in L$ with frequency $\geq$ min_sup

1. Initialize level d := 1
2. Initialize the set of candidate queries Q_1 := ?-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 the frequency of all queries $Q \in Q_d$
   b. Move the queries with frequency $\leq$ min_sup 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. d := d+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 *theta*-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 min_sup 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 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.

As in APRIORI, two evaluation metrics, support and confidence, are used in WARMR. The support of a rule "A ← B" is the probability of both A and B occurring in the records of a database table. The confidence is the probability of occurring A and B over the probability of B. The solution rule must have support and confidence values above the user-defined threshold values.

For the example data set, the candidate generation and evaluation step for the first level (by default, daughter(A,B) is added to items) run as follows (min_sup=0.8):

parent(A, B), daughter(A, B) frequency=0.0 (infreq)
parent(B, A), daughter(A, B) frequency=1.0 (freq)
parent(A, A), daughter(A, B) frequency=0.0 (infreq)
parent(B, B), daughter(A, B) frequency=0.0 (infreq)
parent(A, C), daughter(A, B) frequency=0.0 (infreq)
parent(C, A), daughter(A, B) frequency=1.0 (freq)
parent(B, C), daughter(A, B) frequency=1.0 (freq)
parent(C, B), daughter(A, B) frequency=0.0 (infreq)
female(A), daughter(A, B) frequency=1.0 (freq)
female(B), daughter(A, B) frequency=0.5 (infreq)
female(C), daughter(A, B) frequency=1.0 (freq)

parent(A, B), parent(A, A), parent(B, B), parent(A, C), parent(C, B), parent(B, B), female(B) are found as infrequent and put into I set. parent(B, A), parent(C, A), parent(B, C), female(A), female(C) are found as frequent and put into F set. In the next level, combinations of items in F are evaluated to generate level 2 item-sets.

At the end, one of the frequent queries has items daughter(A, B), parent(B, A) and female(A) which have frequency=1. The following rule can be found as strong:

daughter(A, B) ← parent(B, A), female(A). (frequency=1.0, confidence=1.0)

3.3.7 SAHILP

One major difficulty in ILP is to manage the search space. The most common approach is to perform a search of hypotheses that are local optima for the quality measure. To overcome

this problem, simulated annealing algorithms [1] can be used.

A large number of ILP-based systems use the covering approach in their algorithms. SAHILP [91] uses simulated annealing methods instead of covering approach in ILP for inducing hypothesis. It uses the neighborhood notion in search space, a refinement operator similar to FOIL and weighted relative accuracy [54] (WRAcc) as the quality measure.

WRAcc(H ← B) = p(B) * (P(H | B) - p(H))

For the example data set, the search starts with the most general clause "C_1=daughter(A, B) ← ". According to neighbourhood definition and refinement operator in FOIL, the sample clauses generated in FOIL are also generated in SAHILP.

In the next step, the neighbours (both upward and downward) of the sample clauses are searched and WRAcc values for each clause are calculated. The clause having the maximum WRAcc value will be selected for the best clause.

For the example clause, C = daughter(A, B) ← parent(B, A), female(A);

p(parent(B,A),female(A)) = 2/9

p(daughter(A,B)) = 2/6

p(daughter(A,B) | parent(B,A),female(A)) = 2/2

WRAcc(C) = 2/9 * (1- 2/6) = 4/27

is the maximum WRAcc value in the search space and is selected for the beast clause.

The possibilistic version of SAHILP, namely PosILP, extends propositional logic to the first-order case to deal with exceptions in a multiclass problem [90]. It reformulated the ILP problem in first-order possibilistic logic and redefines the ILP problem as an optimization problem. At the end, it learns a set of prioritized rules.

3.4 Comparison with Proposed Techniques

In this section, we compare the aforementioned ILP systems in terms of basic techniques they employ and some basic features in concept discovery. In comparison, we included the features of search direction, use of mode declaration, use of negative data and handling recursive rules. Although this list can be extended with additional features, we aimed to limit our focus with the ones that we believed to important for facility and quality of concept discovery. The comparison between the proposed methods and the other well-known ILP systems is presented in Table 3.6.

Search direction of the systems is either top-down or bottom-up. In top-down systems, the

Table 3.6: The comparison of ILP systems

System	Search Direction	Basic Technique	Use of Mode Dec.	Use of Neg. Data	Allow Rec.
LINUS	Top-down	Attribute-Value Learning	No	No	No
GOLEM	Bottom-up	Rlgg	Yes	Yes	No
CIGOL	Bottom-up	Inverse Resolution	No	No	No
MIS	Top-down	Refinement Graph	Yes	Yes	Yes
FOIL	Top-down	Refinement Graph	Yes	Yes	Yes
PROGOL	Top-down	Inverse Entailment	Yes	Yes	Yes
WARMR	Top-down	APRIORI	Yes	Yes	Yes
SAHILP	Top-down	Simulated Annealing	Yes	Yes	Yes
C^2D	Hybrid	Absorption, APRIORI Confidence Increase	No	No	Yes
CRIS	Hybrid	Absorption, APRIORI Confidence Increase	No	No	Yes

search starts with a general clause and at each turn, it makes the clause more specific until it covers no negative examples. On the other hand, in bottom-up systems, the search starts with a specialized clause, and at each turn, it generalizes the clause so that it covers more positive examples, until no more improvement is possible. In C^2D an CRIS [41, 40, 44, 42, 43], a hybrid search strategy is employed, such that, the search starts with a bottom-up strategy, starting from target instance, it generates general concept rules having a single predicate in the body, and then it specializes this general rule iteratively until no more improvement is possible on the rule. By this way, general rules that are most relevant to the target are generated and top-down stage starts with them, instead of the empty rule.

The presented systems employ different basic techniques. The search direction is an important factor that is directly related with the selection of the technique. Bottom-up systems, GOLEM and CIGOL, use generalization methods. While GOLEM uses rlgg, CIGOL employs inverse resolution. For top-down systems, refinement graph and APRIORI are the basic methods used for specialization of the clauses. MIS and FOIL use refinement graph, whereas PROGOL works with inverse entailment on the refinement graph. By using APRIORI for generating more specialized clauses, WARMR differs from these systems. It uses support and confidence values for pruning the clause specialization search space. Although it is listed as a top-down system, it is best to emphasize that LINUS is basically an attribute-value learning system. It may be best to describe it as a pioneer of ILP systems. As hybrid systems, the

proposed techniques employ both generalization and specialization methods. Generalization is performed by absorption, which is an inverse resolution operation; and specialization is provided by APRIORI. In this respect, they are similar to WARMR. On the other hand, being a predictive system, they differ from WARMR. In addition to APRIORI, they use a new technique that checks the increase in confidence values, in order the prune the search space.

Use of mode declaration is an effective feature in concept discovery. GOLEM, MIS, FOIL, PROGOL and WARMR use mode declarations for defining structural biases. Although, when it is defined correctly, it may prune search space efficiently, most of the time, it is not straightforward to define modes and the user may need to try out a long list of mode declaration alternatives. Unlike PROGOL and WARMR, C^2D and CRIS do not need input/output mode declarations. They only require type specifications of the arguments, which already exist together with relational tables corresponding to predicates.

Use of negative target examples is another effective tool for search space pruning. However, basic drawback of using negative data is that it may not be available, especially for large data sets stored in the database. GOLEM, MIS, FOIL, PROGOL and WARMR uses negative target instances in concept discovery. In FOIL, if the negative instances are not provided, they can be inferred from Close World Assumption (CWA). On the other hand, the proposed algorithms directly run on relational databases that have only positive data.

Sometimes, recursive clauses are part of the hypothesis in some learning problem sets such as same-generation and ancestor. Recursion is allowed in the proposed methods; however, in PROGOL and WARMR it is very complex to define the correct mode declarations to find the recursive clauses in the hypothesis set. In an experiment on same-generation data set, the proposed algorithms find the correct solution. However, GOLEM, PROGOL and WARMR can not find the correct hypothesis with several mode declarations.

In some cases describing concepts using only background predicates may not be possible or may be very difficult. Recently, new concept discovery systems started to investigate other alternative ways to extend the rules for concept description. The most natural extension is the aggregate predicates. In contrary to aforementioned ILP systems, C^2D and CRIS define and use aggregate predicates in first-order logic, which is described in Section 5.3 and Section 6.2.

CHAPTER 4

AGGREGATE PREDICATES IN CONCEPT DISCOVERY

In this section, we present the use of aggregate predicates in multi-relational learning systems. Firstly we explain the *Mutagenesis* database described in [98]. Then, the usage of aggregate functions over the *Mutagenesis* database is presented. Following this, we give brief information about a popular relational decision tree learning system called Multi-Relational Decision Tree Learning (MRDTL) [59, 7] and the usage of aggregate predicates by extending MRDTL as expressed in [48, 49]. Finally, some other similar systems which supports aggregate predicates are introduced.

4.1 Mutagenesis Database

We have chosen to study the mutagenicity of 230 compounds listed in [98]. Actually, the original format of this database was Prolog syntax. Therefore, the first step in order to use the data set in this work was to translate it to relational format and store it in a relational database. The database consists of two data sets. One of them is the *regression-friendly* data set which has 188 compounds. The other one consists of 42 "regression-unfriendly" compounds. We use the first data set in our experiments. The database consists of 26 tables, of which three tables directly describe the graphical structure of the molecule (*molecule*, *atom* and *bond*). The relationship between these three entities are shown in Figure 4.1. The remaining 23 tables describe the occurrance of predefined functional groups, such as benzene rings.

The database contains descriptions of molecules and the characteristic to be predicted is their mutagenic activity (ability to cause DNA to mutate) represented by the attribute *bool* in the *molecule* table [59]. This problem comes from the field of organic chemistry and the compounds analyzed are nitroaromatics. These compounds occur in automobile exhaust fumes and sometimes are intermediates in the synthesis of thousands of industrial compounds.

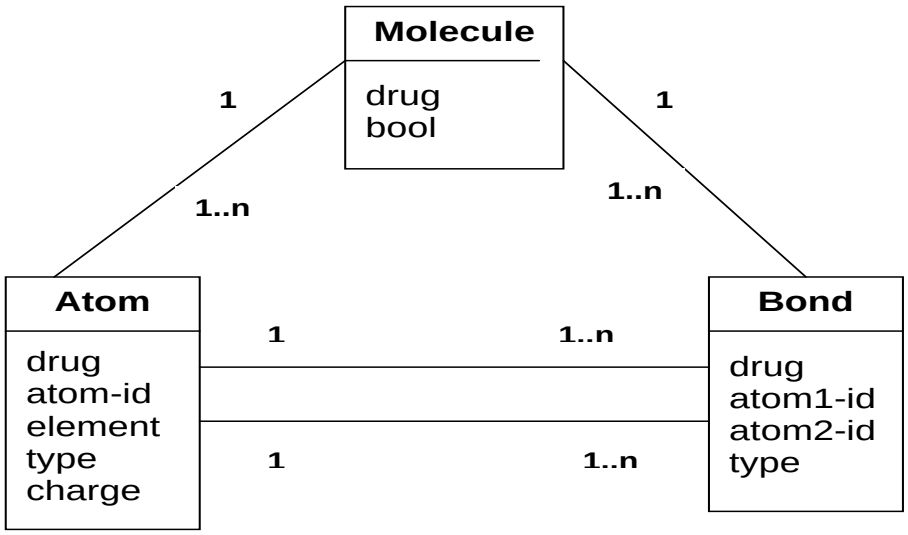

Figure 4.1: The relationship between molecule, atom and bond entities.

High mutagenic level has been found to be carcinogenic.

A recent study using this database [94] recognizes five levels of background knowledge for mutagenesis. Table 4.1 shows the five sets of background knowledge where $B_i \subset B_i + 1$ for i=0..3. In this thesis, we use B_2 in the experiments. Atom-bond descriptions, numeric inequalities, ind-log-lumo definitions for drugs are given in the database.

In the database, the *molecule* (target) relation has 188 records. As there are 230 different drugs in the database, the type table *drug* has 230 records. The type table *bool* has 2 records (true/false). The *atom* relation has 5894 and the *bond* relation 6309 records. There is a one-to-many relationship between *molecule* and *atom* relations over the *drug* argument. A similar relation exists between the *molecule* and *bond* tables. In addition, there is a one-to-many relationship between *atom* and *bond* over the *atomid* argument.

4.2 Aggregate Predicates

Concept discovery aims at finding the rules that best describe the given target predicate (i.e., the concept). An important feature for a concept discovery method is the ability of incorporating aggregated information into the concept discovery. Such information becomes descriptive as in the example ”the total charge on a compound is descriptive for the usefulness or harm-

Table 4.1: Background knowledge for mutagenesis data set in different levels

BACKGROUND	**DESCRIPTION**
B0	Consists of those data obtained with the molecular modeling package QUANTA. For each compound, it obtains the atoms, bonds, bond types, atom types, and partial charges on atoms.
B1	Consists of definitions in *B0* plus indicators *ind1* and *inda* in the database.
B2	The *logp* and *lumo* information for each molecule are added to definitions in *B1*.
B3	Generic 2-D structures, such as methyl groups, nitrogroups, etc., are added to *B2*.
B4	Using 3-dimensional position of each atom in a molecule, generic 3-D calculations are added to *B3*.

fulness of the compound". Therefore, concept discovery system needs aggregation capability in order to construct high quality (with high accuracy and coverage) for such domains.

In relational database queries, aggregate functions characterize groups of records gathered around a common property. In concept discovery, aggregate functions can be utilized in order to construct aggregate predicates that capture some aggregate information over one-to-many relationships. Such relationships are common in databases. In concept discovery, conditions on aggregation such as $count < 10$ or $sum > 100$ may define the basic characteristic of a given concept better. For this reason, in this thesis, we extend the background knowledge with aggregate predicates in order to characterize the structural information that is stored in tables and associations between them.

A brief definition for the popular aggregate functions are given in the following list.

1. SQL provides the *COUNT* function to retrieve the number of records in a table that meets given criteria. We can use the COUNT(*) syntax alone to retrieve the number of rows in a table. Alternatively, a WHERE clause can be included to restrict the counting to specific records.

2. The *MAX* function returns the largest value in a given data series. We can provide the function with a field name to return the largest value for a given field in a table. MAX() can also be used with expressions and GROUP BY clauses for enhanced functionality.

3. The *MIN* function works in the same manner as MAX(), but returns the minimum value for the expression.

4. The *SUM* function is used within a SELECT statement and, predictably, returns the summation of a series of values.

5. The *AVG* function works in a similar manner as SUM() to provide the mathematical average of a series of values.

Property 4.1 *MAX, MIN, SUM and AVG functions can only be applied to arguments having numerical values. The COUNT function can be applied to both numerical and nominal values.*

To characterize the one-to-many relationships that are stored in tables, aggregate predicates are defined and used in the proposed methods, which use aggregate functions.

Aggregate predicates have numeric attributes by their nature. Therefore, in order to add aggregate predicates into the system numeric attribute types should also be handled. Since it is not useful and feasible to define concepts on specific numeric values, comparison operators containing numeric attributes must be considered in concept discovery.

4.3 Multi-Relational Decision Tree Learning

Handling aggregation is more common in relational classification task than concept learning. Therefore, in this section, we present a relational decision tree learning algorithm, namely MRDTL.

Multi-relational patterns can be expressed by using a graphical language consisting of *selection graphs*. These graphs can be translated into SQL or first-order logic expressions [48, 49].

Definition 4.1 *A* Selection Graph (SG) G *is a directed graph* (N, E)*, where* N *is a set of triples (T, C, s) called* selection nodes*, T is a table in the data model and C is a, possibly empty set of conditions on attributes in T of type T.A* $\oplus$ *c;* $\oplus$ *is one of the usual selection operators* $=$, $\leq$, $\geq$, *etc. s is a flag with possible values* open *and* closed *[59, 7].*

E is a set of tuples (p, q, a, e) called *selection edges*, where *p* and *q* are selection nodes and *a* is an association between *p.T* and *q.T* in the data model. *e* is a flag with possible values *present* and *absent*. The selection graph contains at least one node n_0 that corresponds to the target table T_0.

Selection graphs can be represented as directed labeled graphs. An example is shown in Figure 4.2 [59] based on the data model shown in Figure 4.1. The current graph selects those

molecules that have at least one atom whose partial charge is less than or equal to -0.392, but for which none of them have charge less than or equal to -0.392 and element equal to 'b' at the same time.

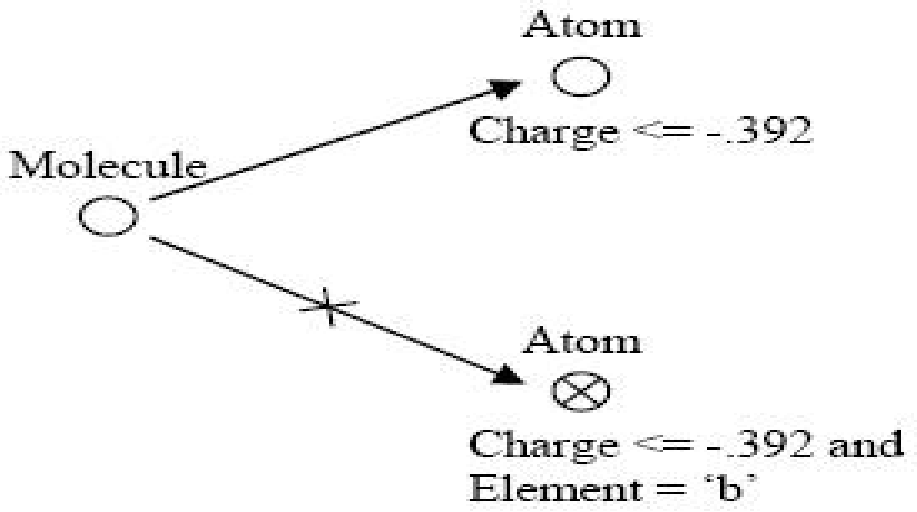

Figure 4.2: An example Selection Graph for mutagenesis data set

Note, from the figure, that the value of s is represented by the presence or absence of a cross in the node, representing the value open and closed respectively. The value for e, in turn, is indicated by the presence or absence of a cross on the corresponding arrow representing the edge.

A present edge between *p* and *q* represents a join between these two tables using as the primary key in *p* and foreign key in *q*. The edge combined with a list of conditions selects those records that match the list of conditions and belong to the join. On the other hand, an absent edge between table *p* and *q* combined with a list of conditions selects those records in *p* that do not match the list of condition. Any subgraph that is pointed by an absent edge thus corresponds to a set of negative conditions.

An important system related to proposed algorithms is Multi-Relational Decision Tree Learning (MRDTL). MRDTL constructs *Selection Graph* [59] for rule discovery. It is an extension of the logical decision tree induction algorithm called TILDE proposed by Blockeel [8]. TILDE uses first-order logic clauses to represent decisions (nodes) in the tree. The data are represented in first-order logic rather than a collection of records in a relational database. MRDTL extends TILDE's approach to deal with records in relational databases. They use

similar refinement operators, and the way the tree is inducted follows the same logic. For more information about TILDE refer to [8].

MRDTL adds decision nodes to the tree through a process of successive refinement until some termination criteria is met. The choice of refinement at each step is guided by a suitable measure (e.g. information gain). MRDTL starts with a single node at the root of the tree, which represents the set of all objects of interest in the target table T_0. For detailed explanation of MRDTL algorithm, refer [59].

Afterwards, a more efficient version of MRDTL which is called MRDTL2 [36] was developed. MRDTL has two significant limitations: slow running time and inability to handle missing attribute values. MRDTL-2 includes some techniques to overcome these limitations. For more information about MRDTL-2 refer to [7].

Leiva [59] presents a number of experiments using MRDTL. One of them involves the popular data set "Mutagenesis" given in Section 4.1. MRDTL has 88 percent accuracy as the best result in the experiments.

MRDTL inspired this thesis for defining and using aggregation, however, we followed a logic-based approach and included aggregate predicates in an ILP-based context for concept discovery.

4.4 Using Aggregate Predicates in MRDTL

Knobbe extended the structure of selection graphs by adding the possibility of aggregate conditions, resulting in *generalised selection graphs* (GSG) [48, 49].

Definition 4.2 *An* aggregate condition *is a triple (f, θ, v) where* f *is an aggregate function, θ a comparison operator, and* v *a value of the codomain of* f.

Definition 4.3 *A* Generalised Selection Graph (GSG) *is a directed graph (N, E), where N is a set of triples (t, C, s), t is a table in the data set and C is a, possibly empty, set of conditions on attributes in t of type (t.a θ c); θ one of the following operators, $<$, $\leq$, $\geq$. The flag* s *has the possible values* open *and* closed. E *is a set of tuples (p, q, a, F) where p, q $\in$ N,* a *is an association between* p.t *and* q.t *in the data set, and* F *is an aggregation condition. The generalised selection graph contains at least one root node (represents target table).*

Briefly, generalised selection graphs are are supersets of generalised graphs. More information about GSGs are given in [48].

The *Mutagenesis* data set (given in Section 4.1) is tested with the GSG structure. An example graph found after the execution of the algorithm is given in Figure 4.3.

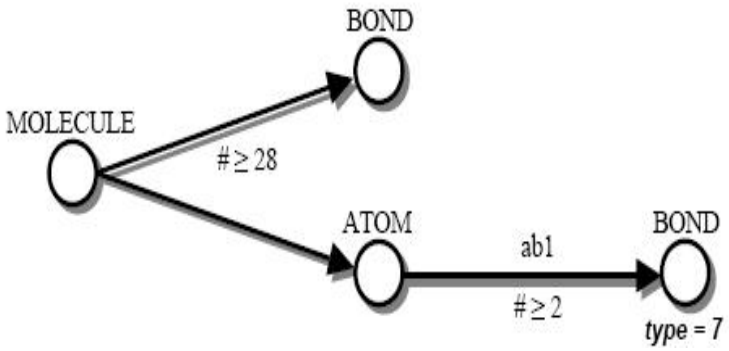

Figure 4.3: An example Generalised Selection Graph for mutagenesis data set

If a drug has more than 28 (or equal) bonds and has an atom which exists in more than (or equal) two bonds having type 7, then it is mutagenic.

The best accuracy in the experiment results is 0.948. The details of the results are given in [48].

4.5 Aggregate Predicates in Similar Systems

There are some other works that uses aggregation in multi-relational learning.

Crossmine [111] is such an ILP based multi-relational classifier that uses TupleID propagation and a selective sampling method. In multi-relational classification, there is one target relation R, and each tuple in R (target tuples) is associated with a label. Many ILP approaches takes one or more joins between R and the non-target relations which is expensive in both time and space.

In Crossmine, the basic idea is to propagate the tuple IDs (together with their associated class labels) in the target relation to other relations. In the relation to which the IDs are propagated, each tuple *t* is associated with a set of IDs, which represent the target tuples that are joinable with *t*. Besides propagating IDs from target relation to non-target relations, one can propagate the IDs transitively to additional non-target relations to search for good predicates among many relations. The idea of tuple ID propagation is to virtually join the relations with minimal cost, and then find good predicates in the joined relation. CrossMine

obtains high scalability by avoiding the high cost of physical joins.

Multi-relational g-mean decision tree, called Mr.G-Tree [58] is proposed to extend the concepts of propogation described in Crossmine by introducing the g-mean Tuple ID propagation algorithm, also known as GTIP algorithm. GTIP maintains the original data distribution in each non-target relation by restoring the number of target classes of each tuple to a single one. It also uses the primary and foreign key properties in the database for performing joins in the algorithm.

Classification with Aggregation of Multiple Features (CLAMF) extends TupleID propagation in order to efficiently perform single and multi-feature aggregation over related tables [32]. It classifies multi-relational data using aggregation involving single and multiple features without physical joins of the data. It supports several aggregate functions for different numbers of features and data types.

Traditional tree learning algorithms assume that instances in the training data are homogenous and independently distributed. Relational probability trees (RPTs) extend standard probability estimation trees to a relational setting in which data instances are heterogeneous and interdependent. RPT models estimate probability distributions over possible attribute values. In [76], the algorithm for learning the structure and parameters of an RPT searches over a space of relational features that use aggregation functions (e.g. AVERAGE, MODE, COUNT) to dynamically propositionalize relational data and create binary splits within the RPT.

A hieararcy of relational concept classes in order of increasing complexity is presented in [80], where the complexity depends on that of any aggregate functions used. It also presents an overview of existing target-dependent aggregation methods and their limitations on a relational business domain.

An approach based on random forests is presented in [6], in which aggregation and selection are combined efficiently. Random forest induction is a bagging method that builds decision trees using only a random subset of the feature set at each node. In that approach, the decision trees that are constructed contain tests with first order logic queries that may involve aggregate functions. The argument of these aggregate functions may again be a first order logic query.

CHAPTER 5

C^2D: CONFIDENCE-BASED CONCEPT DISCOVERY

A concept is a set of frequent patterns, embedded in the features of the concept instances and relations of objects belong to the concept with other objects as defined in Section 2.1.1. C^2D [41, 40, 42, 44] is a concept discovery system that uses first-order logic as the concept definition language and generates a set of concept rules having the target concept in the head. Since the predicate calculus is capable of representation of relations between objects via predicates and relations between predicates via shared variables among predicate arguments, in this thesis, first-order logical framework is chosen as the concept definition language where concepts/relational patterns are represented by function-free concept rules.

In the proposed algorithms, a concept rule can be interpreted as a partial definition for a concept, where the head predicate identifies the defined concept and the predicates in the body of the rule represent the required features and relations for an object that belongs to this concept.

The user supplies the type declarations of predicate arguments via C^2D's graphical user interface given in Figure 5.1. The user also supplies the concept to be learned via the interface. In the graphical user interface, first of all, the user selects the data set from the combo box, then the parameters (min_sup, min_conf, B, max_depth) are defined. The "Allow Recursion" checkbox is used to enable recursive rule search in C^2D and CRIS. However, "Consider Only Related Facts" checkbox is used only for C^2D to enable searching transitive rules. When the C^2D button is pressed, the C^2D algorithm starts. On the other hand, when the CRIS button is pressed, the CRIS algorithm starts. The details of the implementation are given in Appendix A.

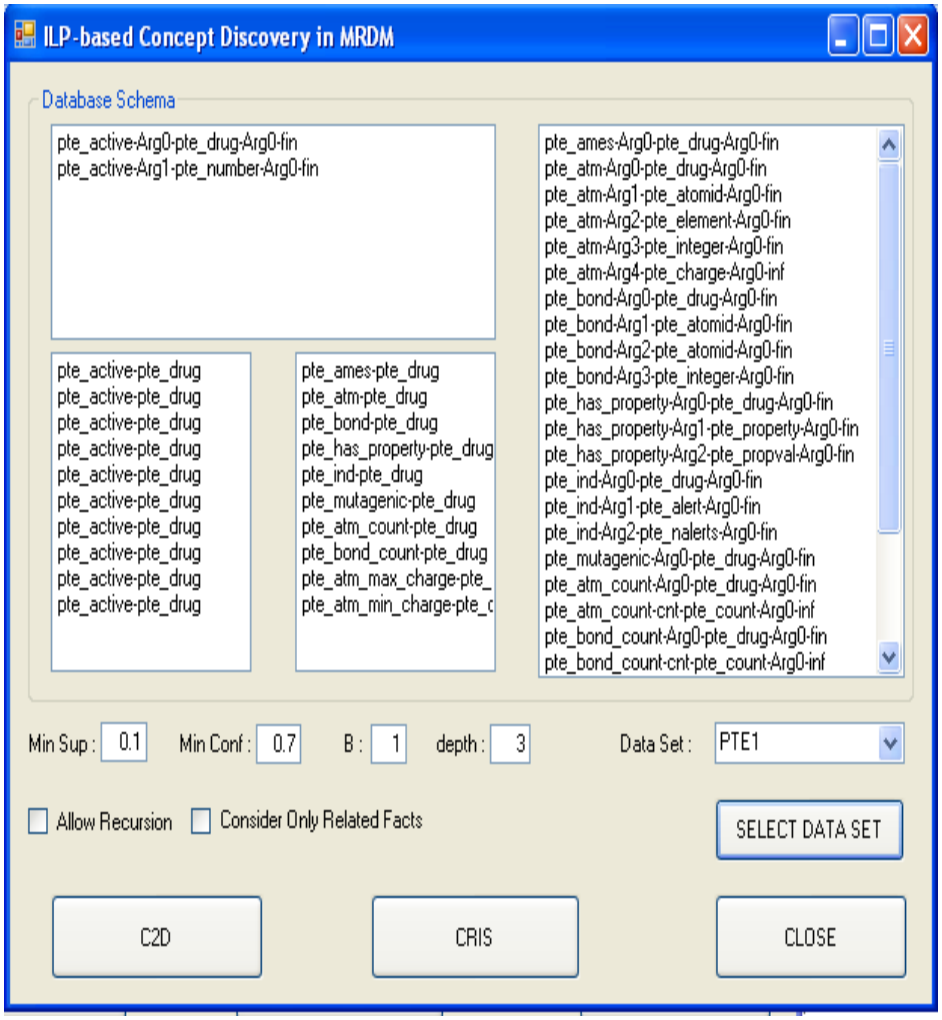

Figure 5.1: The graphical user interface of C^2D

5.1 Support and Confidence

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

The detailed information about the frequency and confidence definitions for query, query extension and clause were given in [17, 60].

The support and confidence definitions for association rules having one head relation,

which are called as *concept rule*, are given in Section 2.1.

In the literature, the *support* and *confidence* values of a concept rule are obtained with the SQL queries given in [20]. These queries are shown in Table 5.1 and Table 5.2.

Table 5.1: The SQL queries for support calculation

Support = COUNT1 / COUNT2
COUNT1:
SELECT COUNT(*) FROM
    SELECT DISTINCT(key fields in the head predicate)
    FROM relations in C
    WHERE conditions expressed in the rule C
COUNT2:
SELECT COUNT(*) FROM
    SELECT DISTINCT(key fields in the head predicate)
    FROM the head relation in C

Table 5.2: The SQL queries for confidence calculation

Confidence = COUNT3 / COUNT4
COUNT3:
SELECT COUNT(*) FROM
    SELECT DISTINCT(key fields in the body predicates
        that are bound in the head predicate)
    FROM relations in C
    WHERE conditions expressed in the rule C
COUNT4:
SELECT COUNT(*) FROM
    SELECT DISTINCT(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

For the *daughter* example, the SQL queries for the below rule are shown in Table 5.3 and Table 5.4:

daughter(A, B) ← parent(B, A).

In this thesis, we use the definition of *support* query as given in [20], however the application of the *confidence* query is modified since the current definition causes problems for the

Table 5.3: The SQL queries for support of daughter(A, B) ← parent(B, A)

```
Support = COUNT1 / COUNT2
COUNT1:
SELECT COUNT(*) FROM
        SELECT DISTINCT(d.arg1, d.arg2)
        FROM daughter d, parent p
        WHERE d.arg1=p.arg2 AND d.arg2=p.arg1
COUNT2:
SELECT COUNT(*) FROM
        SELECT DISTINCT(d.arg1, d.arg2)
        FROM daughter d
```

Table 5.4: The SQL queries for confidence of daughter(A, B) ← parent(B, A)

```
Confidence = COUNT3 / COUNT4
COUNT3:
SELECT COUNT(*) FROM
        SELECT DISTINCT(p.arg1, p.arg2)
        FROM daughter d, parent p
        WHERE d.arg1=p.arg2 AND d.arg2=p.arg1
COUNT4:
SELECT COUNT(*) FROM
        SELECT DISTINCT(p.arg1, p.arg2)
        FROM parent p
```

cases where the head relation has variables that do not exist in the body. In order to illustrate the problem with the classical *confidence* query, consider the following example rules:

daughter(A, B) ← parent(B, tom). (s=0.5, c=1.0)

daughter(A, B) ← female(A). (s=1.0, c=0.67)

Confidence is the ratio of number of positive instances deducible from the concept rule over number of examples deducible from the rule. In other words, it shows how strong the rule is. For the first rule, the *confidence* value shows that it is very strong. However, out of the following four deducible facts *daughter(ann, ann)*, *daughter(mary, ann)*, *daughter(tom, ann)* and *daughter(eve, ann)*, only one of them (only daughter(mary, ann) is positive) exists in the database. As a result, the first rule covers some negative instances.

Similarly, for the second rule, the *confidence* value is also high. The following facts *daughter(ann, ann)*, *daughter(ann, mary)*, *daughter(ann, tom)*, *daughter(ann, eve)*, *daughter(mary, ann)*, *daughter(mary, mary)*, *daughter(mary, tom)*, *daughter(mary, eve)*, *daugh-*

ter(eve, ann), daughter(eve, mary), daughter(eve, tom) and *daughter(eve, eve)* are deducible from the second rule, but only 2 of them (only daughter(mary, ann) and daughter(tom, eve) are positive) exist in the database. Out of 12 possible ground instances of predicate *daughter*, only 2 of them are concept instances. The *confidence* of this rule must be very low (such as 2/12 = 0.17).

In order to solve this problem, we add type relations to the body of the rule corresponding to the arguments of the head predicate whose variable does not appear in the body predicates. The type tables for the arguments of the target relation are created in the database (if they do not exist).

For the *daughter* example, *person* table is the type table, which contains all values in the domain of the corresponding argument of the target relation in the database. For the *daughter* example, *person* table contains 4 records which are *ann, mary, tom* and *eve*. Each new literal has a relation name as the corresponding head predicate argument type and has one argument that is the same as the corresponding head predicate argument. The rules obtained by adding type literals are used only to compute the *confidence* values, and for the rest of the computation, original rules without type literals are used.

The addition of type relations models the positive instances better and reflects the confidence value correctly. By this way, negative instances can be deduced as in CWA. Besides this, since the type relation is always true for the instance, this modification does not affect the semantics of the rule. In addition, the definition of the *confidence* query remains intact.

As a result of this modifications, the new concept rules' *support* and *confidence* values for the daughter example are as follows:

daughter(A, B) ← parent(B, tom), person(A). (s=0.5, c=0.25)
daughter(A, B) ← female(A), person(B). (s=1.0, c=0.17)

5.2 The Algorithm

C^2D is developed on the basis of the systems described in [100, 101]. It employs a coverage algorithm in constructing concept definition. In C^2D, four mechanisms are used for pruning the search space.

1. The first one is a generality ordering on the concept rules based on θ-subsumption and is defined in Section 2.1:

Strategy 5.1 *In C^2D, candidate rules are generates according to θ-subsumption (defined in Section 2.1).*

For instance, consider the following two frequent rules from the daughter example (Table 3.1):

C_1: d(A, B) ← p(B, C).

C_2: d(A, B) ← p(B, C), f(C).

As the head of C_1 and C_2 (d(A,B)) are same and body of C_1 is a subset of C_2, C_1 is more general than C_2 and it θ-subsumes C_2.

2. The next pruning strategy is applied in the beginning of the main algorithm.

Strategy 5.2 *C^2D takes only related facts of the selected target concept instance to generate the rules in the generalization phase.*

As an example, for the target concept instance t(a, b), a background relation r (c, d), which has no records having *a* or *b*, is an unrelated relation and is pruned in the generalization step for rule construction.

It is an effective pruning mechanism for eliminating arbitrary rule structures. However, it may miss transitive rules and to handle this problem, an optional built-in function is implemented in the main algorithm. This is explained in Section 5.4.

3. The third pruning strategy is about the use of confidence. For this strategy, first we define a "non-promising rule".

Definition 5.1 *Let C_1 and C_2 be the two parent rules of the concept rule* C *in the Apriori search lattice [3]). If the confidence value of* C *is not higher than the confidence values of C_1 and C_2, then it is called a non-promising rule.*

Strategy 5.3 *In C^2D, non-promising rules are pruned in the search space.*

By this way, in the solution path, each specialized rule has higher confidence value than its parents. A similar approach is used in the Dense-Miner system [85] for traditional association rule mining.

For the illustration of this technique on the *daughter* example, consider the following two frequent rules in the first level of the APRIORI lattice:

C_1: d(A, B) ← p(B, C). (c=0.25)

C_2: d(A, B) ← f(C). (c=0.125)

These rules are suitable for union since their head literals are same and they have exactly one different literal from each other. Possible union rules are as follows:

C_3: d(A, B) ← p(B, C), f(C). (s=1, c=0.25)

C_4: d(A, B) ← p(B, C), f(D). (s=1, c=0.25)

C_3 and C_4 are frequent rules but they do not have higher confidence values than C_1. Therefore, they are pruned in the search space.

4. The last pruning strategy employed in C^2D, which is also a novel approach, utilizes primary-foreign key relationship between the head and body relations:

   Strategy 5.4 *If a primary-foreign key relationship exists between the head and the body predicates, the foreign key argument of the body relation can only have the same variable as the primary key argument of the head predicate in the generalization step.*

   For example, in the Mutagenesis database, the first concept example is molecule(d1, true) and a related fact is atom(d1, d1_1, c, 22, -0.117). As there is a primary-foreign key relationship between molecule and atom relations through the "drug" argument (first argument), some of the example rules obtained after the generalization step are as follows:

   molecule(A, true) ← atom(A, d1_1, c, 22, -0.117).
   molecule(A, true) ← atom(B, d1_1, c, 22, -0.117).
   molecule(A, true) ← atom(d1, B, c, 22, -0.117).
   ...
   molecule(A, true) ← atom(A, B, c, 22, -0.117).
   molecule(A, true) ← atom(A, d1_1, B, 22, -0.117).
   ...

   On the basis of this idea, in the generalization step, rules that have different attribute variables for primary-foreign key attributes are not allowed. For example, the rule

"molecule(A, true) ← atom(B, C, c, 22, -0.117)." is not generated in the generalization step.

Another new feature of C^2D is its parametric use for *support, confidence, recursion* and *f-metric*. The user can set support threshold and she/he can allow or disallow the use of support as a part of the pruning mechanism. It is possible to set confidence threshold for selecting the best rule, so that the best rule will have an acceptable confidence value. Similarly, by changing the value of the recursion parameter, it is possible to allow generating recursive or only linearly recursive hypothesis, or totally disallow recursive concept definitions. Another parametric declaration is for the *f-metric* (adapted from f-score formula [35]), whose definition is as follows:

Definition 5.2 $f - metric = \frac{(B+1)\times confidence\times support}{(B\times confidence)+support}$

The user can emphasize the effect of support or confidence by changing the value of B. If the user defines B to be greater than 1, then confidence has a higher effect. On the other hand, if B has a value less than 1, then support has a higher effect. Otherwise, both support and confidence have equivalent weight in the evaluation. An overview of quality measures is given in [54].

The algorithm of C^2D [41, 40, 42, 44], given in Table 5.5, starts with selecting a positive concept instance. The most general rules with two literals, one in the head and one in the body, that entail the positive example are generated and then the concept rule space is searched with an APRIORI-based specialization operator. In the refinement graph, if support parameter is on and the frequency of a rule is below the support threshold, it is pruned as an infrequent rule. In addition to this, rules whose confidence values are not higher than their parents' confidence values are also eliminated. When the maximum depth reached or no more candidate rule can be found, if confidence parameter is on, then the rules that have less confidence value than the confidence threshold are eliminated for the solution set. Among the produced strong and frequent rules, the best rule (having highest f-metric value) is selected and the rule search is repeated for the remaining concept instances that are not in the coverage of the selected hypothesis rules. If there is no possible best rule found for the selected positive concept instance, then the algorithm will select another positive concept instance and start from the beginning. At the end, some uncovered positive concept instances may exist because of the user settings for the thresholds. In the rest of this section, the main steps of the algorithm are described.

Table 5.5: C^2D algorithm

- **Input:** *I*: concept instance set (from DB), *BF*: background facts
- **Output:** *H*: hypothesis set (initially *H*=∅)
- **Parameters:** *min-sup*: support threshold, *min-conf*: confidence threshold
  B (used in f-metric), *md*: maximum depth (body literal count)
- **Local Variables:** C_i: candidate rules set at level i, *d*: level
  G: generalization rules set, *FSC*: frequent and strong rules set
- Repeat until *I* is covered by *H* (until $I = \emptyset$) OR
  No more possible *G* can be found according to given parameters:
  1. Select p from *I*.
  2. GENERALIZATION: Generate *G* of p by using *BF*
  3. Initialize C_1:=*G*, *FSC*:=∅, *d*:=1
  4. REFINEMENT OF GENERALIZATION:
     While $C_d \neq \emptyset$ and $d \leq md$
     a. FSC = FSC ∪ FREQUENT_STRONG_RULES(C_d, min-sup)
     b. C_{d+1} = CANDIDATE_GENERATION(FSC, min-sup)
        i. UNION: For each pair of the rules in level *d*
           compute each possible union rule.
        ii. For each union rule satisfying *min-sup*;
           -SPECIALIZATION: Generate rules by unifying existential variables
           -FILTERING: Discard non-frequent and less-confident rules
     c. *d:=d+1*.
  5. EVALUATION:
     Eliminate rules from *FSC* according to *min-conf*
     Select c_{best} from *FSC* according to selected evaluation criteria
  6. COVERING: Compute $I_c \subseteq I$ covered by c_{best}
  7. $H := H \cup c_{best}$
  8. $I := I - I_c$
- Return *H*.

Generalization: In the generalization step, after picking a positive example, C^2D 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 value 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). It utilizes the absorption operator 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 rule and the related fact, RF, on the other arm of the tree, as follows: $CD = (CI \cup \neg RF)\theta_2^{-1}$ For generalizations all possible values of inverse substitution θ_2^{-1} must be searched.

Concept instance and each related fact are generalized into two literal rules, 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 generalizations are obtained.

In the daughter example, C^2D 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 all concept instances together with all related facts. For instance, by applying absorption operator to the concept instance "daughter(mary, ann)" and to the related fact "parent(ann,mary)", the concept descriptions of the form

$$\{daughter(mary, ann) \leftarrow parent(ann, mary)\}\theta_2^{-1}$$

are derived. Table 5.6 consists of the possible inverse substitutions and the resultant concept descriptions. In the right-hand column of the table, the inverse substitutions are in the form of (term, the locations of the term in the rule)/variable where the locations of the term in the rule are represented with 2-tuples such that first entry represents the order of the predicate in the rule and the second entry shows the argument order in the predicate. (Both predicate and argument order start with 0)

To handle indirectly related relations in C^2D, an option called *add indirectly related facts* is implemented in the main algorithm.

Strategy 5.5 *The add indirectly related facts option adds the facts, which are related with the related facts of the selected target instance, into APRIORI lattice. In other words, indirectly related facts of the selected target instance are added to search space to find transitive rules.*

The details of using indirectly related facts are described in Section 5.4.

After the generalization phase, C^2D populates first level of the APRIORI lattice with these two literal concept descriptions obtained in the generalization phase.

In the *daughter* example, for the concept instance *daughter(mary, ann)*, two literal generalizations are generated in the presence of related facts and the first level of the search lattice is populated with these generalizations. With the support threshold value 0.8, the system eliminates the infrequent rules. Among 18 rules generated for *daughter(mary, ann)* and *parent(ann, mary)*, only 6 rules (shown in bold) satisfy the threshold. The relative support values of two literal candidate rules are given in Table 5.7.

Table 5.6: Two predicate concept descriptions generated by C^2D

Concept Description	**Value of θ_2^{-1}**
d(A, ann) ← p(ann, A)	{(mary, [0, 0][1, 1])/A}
d(A, ann) ← p(ann, B)	{(mary, [0, 0])/A, (mary, [1, 1])/B}
d(mary, A) ← p(A, mary)	{(ann, [0, 1][1, 0])/A}
d(mary, A) ← p(B, mary)	{(ann, [0, 1])/A, (ann, [1, 0])/B}
d(A, ann) ← p(B, mary)	{(mary, [0, 0])/A, (ann, [1, 0])/B}
d(A, B) ← p(B, mary)	{(mary, [0, 0])/A, (ann, [0, 1][1, 0])/B}
d(A, B) ← p(C, mary)	{(mary, [0, 0])/A, (ann, [0, 1])/B, (ann, [1, 0])/C}
d(mary, A) ← p(ann, B)	{(mary, [1, 1])/B, (ann, [0, 1])/A}
d(mary, A) ← p(A, B)	{(mary, [1, 1])/B, (ann, [0, 1][1, 0])/A}
d(mary, A) ← p(B, C)	{(mary, [1, 1])/C, (ann, [0, 1])/A, (ann, [1, 0])/B}
d(A, B) ← p(ann, A)	{(mary, [0, 0][1, 1])/A, (ann,[0, 1])/B}
d(A, ann) ← p(B, A)	{(mary, [0, 0][1, 1]/A), (ann, [1, 0])/B}
d(A, B) ← p(B, A)	{(mary, [0, 0][1, 1])/A, (ann, [0, 1][1, 0])/B}
d(A, B) ← p(C, A)	{(mary, [0, 0][1, 1])/A, (ann, [0, 1])/B, (ann, [1, 0])/C}
d(A, B) ← p(ann, C)	{(mary, [0, 0]/A), (mary, [1, 1])/C, (ann,[0, 1])/B}
d(A, ann) ← p(B, C)	{(mary, [0, 0])/A, (mary, [1, 1])/C, (ann, [1, 0])/B}
d(A, B) ← p(B, C)	{(mary, [0, 0])/A, (mary, [1, 1])/C, (ann, [0, 1][1, 0])/B}
d(A, B) ← p(C, D)	{(mary, [0, 0])/A, (mary, [1, 1])/D, (ann, [0, 1])/B, (ann, [1, 0])/C}

Refinement of Generalization: C^2D refines the two literal concept descriptions with an APRIORI-based specialization operator that searches the concept rule 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 rule set selection from candidate rules and candidate rule set generation as refinements of the frequent rules in the previous level. The standard APRIORI search lattice is extended in order to capture first-order logical rules and the candidate generation and frequent pattern selection tasks are customized for first-order logical rules.

In the candidate rule generation step, candidate rules for the next level of the search space are generated. Candidate rule generation is composed of three important steps:

1. Frequent rules of the previous level are joined to generate the candidate rules via union operator. In order to apply the union operator to two frequent concept rules, these rules must

Table 5.7: The relative support values of two literal concept rules generated in C^2D

Concept Description	**Relative Support Value**
d(A, ann) ← p(ann, A)	0.5
d(A, ann) ← p(ann, B)	0.5
d(mary, A) ← p(A, mary)	0.5
d(mary, A) ← p(B, mary)	0.5
d(A, ann) ← p(B, mary)	0.5
d(A, B) ← p(B, mary)	0.5
d(A, B) ← p(C, mary)	**1.0**
d(mary, A) ← p(ann, B)	0.5
d(mary, A) ← p(A, B)	0.5
d(mary, A) ← p(B, C)	0.5
d(A, B) ← p(ann, A)	0.5
d(A, ann) ← p(B, A)	0.5
d(A, B) ← p(B, A)	**1.0**
d(A, B) ← p(C, A)	**1.0**
d(A, B) ← p(ann, C)	**1.0**
d(A, ann) ← p(B, C)	0.5
d(A, B) ← p(B, C)	**1.0**
d(A, B) ← p(C, D)	**1.0**

have the same head literal, and bodies must have all but one literal in common.

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

$$C_1 \cup C_2 = \{C_1 \cup \ l_{21} \mid C_{12} = C_1 \cap C_2\theta \ \wedge \ C_2\theta - C_{12} = l_{21}\}. \tag{5.1}$$

As shown above, the union of two rules is in fact appending the literal in C_2 but not in C_1 to the body of the rule 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 rule C_1. Since there may be more than one intersection of two frequent rules, it is possible to produce multiple unions of two rules.

If the support of the union is above the threshold value and has confidence value larger than parent's values, it will be added to the second level of the search lattice. If the concept descriptions are refined with only union operator, the search space will consist of rules that have body literals directly bound to the head literal through head variables. The structure of such rules can be figured out as in Figure 5.2 [100].

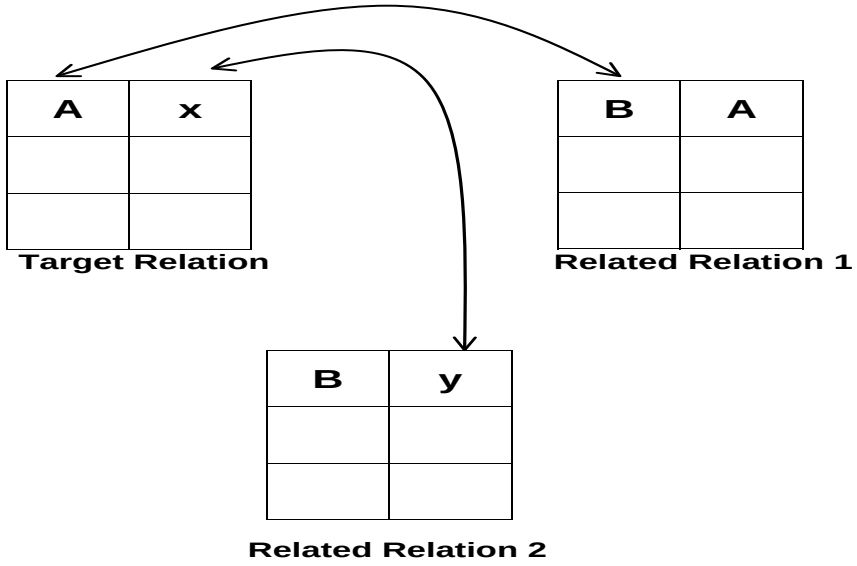

Figure 5.2: A rule with body predicates directly bound to the head predicate

Since only rules that have the same head literal are combined, the search space is partitioned into disjoint APRIORI sub-lattices according to the head literal. In addition, the system does not combine rules that are specializations of the same candidate rule produced in the second step of the candidate rule generation task in order to prevent logical redundancy in the search space. To do this, the system keeps a group number for each rule in the search lattice; the specializations of a rule in the same level of the search lattice have the same group number.

2. For each frequent union rule, a further specialization step is employed that unifies the existential variables of the same type in the body of the rule. By this way, rules with relations indirectly bound to the head predicate can be captured. The structure of such rules can be figured out as in Figure 5.3 [100].

The specialization operator unifies the existential variables of the same type in the body of the rule. For the union rule ($C_1 \cup C_2$), the specialization is done as follows (assume l_{21} is added to C_1):

i. If an existential parameter in l_{21} (which does not exist in the head literal) has the same name but different type with an existential variable in C_1, then the system changes its name (gives a new name)

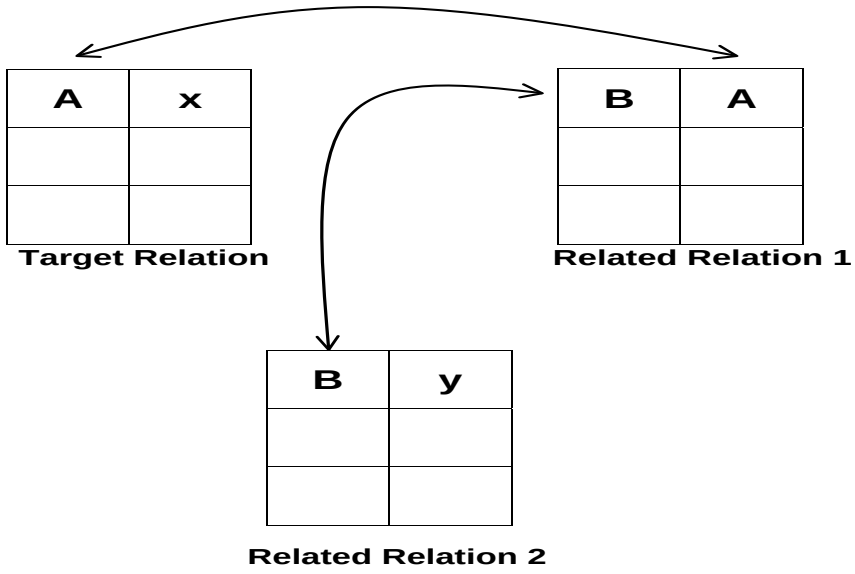

Figure 5.3: A rule with body predicates indirectly bound to the head predicate

ii. The system creates a name list for each existential variable in l_{21}. But that variable must not exist in the head literal and must not exist in the body of C_2 more than once. Otherwise, its name remains same. The list has possible names exist in C_1 which has the same type with that variable of l_{21}. In addition, that list has an extra new name for the variable.

iii. All possible combinations of names in the lists create a specialization of the union rule.

3. Except for the first level, the candidate rules that have confidence value not higher than parent's confidence values are eliminated. If the rule has confidence value as 1, it is not further specialized in the following steps. If the support value is also 1, then it is a solution for the uncovered examples and it is added to the hypothesis set.

For the illustration of this step on the *daughter* example, consider the following two frequent rules in the first level of the lattice:

C_1: daughter(A, B) ← parent(B, C). (c=0.25)

C_2: daughter(A, B) ← female(C). (c=0.125)

These rules are suitable for union since their head literals are same and they have exactly one different literal from each other. Possible union rules are as follows:

C_3: daughter(A, B) ← parent(B, C), female(C). (s=1, c=0.25)

C_4: daughter(A, B) ← parent(B, C), female(D). (s=1, c=0.25)

C_3 and C_4 are frequent rules but they do not have higher confidence values than C_1. Therefore, they are pruned in the search space.

Evaluation: 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 and strong confident candidate rules that induce the current concept instance. Then it eliminates the rules having less confidence value than the confidence threshold. Finally, the system should decide on which rule in the search tree represents a better concept description than other candidates on the basis of an evaluation criterion. As discussed in Section 5.1, this thesis proposes an improved confidence-based evaluation criterion. On this system, several experiments have been conducted to observe the effects of conventional and improved confidence-based evaluation. The results of the experiments are given in Chapter 7.

Coverage: After the best rule is selected, concept instances covered by this rule are determined and removed from the concept instances set. The main iteration continues until all concept instances are covered or no more possible candidate rule can be found for the uncovered concept instances.

In the *daughter* example, the search tree constructed for the instance *daughter(mary, ann)* is traversed for the best rule. Under the *f-metric* evaluation criterion, the rule *daughter(A, B)* ← *parent(B, A), female(A)* with support value of 1.0 and the confidence value of 1.0 (f-metric=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 following hypothesis:

$$daughter(A, B) \leftarrow parent(B, A), female(A).$$

5.3 Aggregate Predicates in C^{2}D

Aggregate functions provide a unique way of characterizing groups of records which are common in databases. To characterize the one-to-many relationships that are stored in tables, aggregate predicates are defined and used in the proposed methods.

Definition 5.3 *An* Aggregate Predicate *(*Π*) is a predicate that defines aggregation over an attribute of a given* Predicate *(*α*). We use a notation similar to given in [33] to represent the general form for* aggregate predicates *as follows:*

$$\Pi^{\alpha;\beta}_{\gamma;\omega}(\gamma, \sigma)$$

where α is the predicate over which the Aggregate Function *(ω) (*COUNT, MIN, MAX, SUM *and* AVG *are the frequently used functions) is computed,* Key *(γ) is a set of arguments that will form the key for* Π, Aggregate Value *(σ) is the value of ω applied to the set of values defined by* Aggregate Variable List *(β).*

Definition 5.4 *An* aggregate rule *is a concept rule which has at least one aggregate predicate in the body relations.*

Definition 5.5 *An* aggregate query *is a SQL statement having* SELECT *and* GROUP BY *commands and aggregate functions defined in SQL. The instances of aggregate predicates are created by using aggregate queries. Given* $\Pi^{\alpha;\beta}_{\gamma;\omega}(\gamma,\sigma)$*, the corresponding aggregate query is*

SELECT γ, $\omega(\beta)$ as σ
FROM α
GROUP BY γ

As an example, Mutagenesis database [98] is used for explaining the usage of aggregate functions. The structure of the target table and two important background relations in the database are shown in Figure 4.1.

$$atom\_count^{atom;atom-id}_{drug;COUNT}(drug, cnt)$$

is an example aggregate predicate that can be defined in the Mutagenesis database. For simplicity, we abbreviate it as atom_count(drug, cnt). Some of the example aggregate predicates that can be defined in the database are:

bond_count(drug, cnt).
atm_bond_count(atom-id, cnt).
charge_max(drug, mx).
charge_min(drug, mn).

An example aggregate rule is:

$$molecule(d1, true) \leftarrow atom\_count(d1, A), A \geq 28.$$

For example, the instances of *atom_count* aggregate predicate on *atom* relation are constructed by the following query:

SELECT drug, COUNT(atom-id) as cnt

FROM atom

GROUP BY drug

Aggregate predicates have numeric attributes by their nature. For this reason, we worked on the representation of numeric attributes in concept description. For the predicates having numerical attributes, it is infeasible to generate rules that test such numeric values through equality. For example, the *atom* relation in the Mutagenesis database has the argument "charge" which has floating-point values. It is infeasible to search for a rule such as: *A drug is mutagenic if it has the charge of -0.117.* As there are many possible values in the relation, such a rule may be eliminated according to minimum support criteria. Instead, to search for drugs which has charge less/greater than some threshold value will be more feasible. For this purpose, the following modification is applied in C^2D for numeric attributes:

As the first step, domain of the numerical attributes are explicitly defined as "infinite" in the generalization step. For the "infinite" attributes, concept rules are generated on the basis of the following strategy.

Strategy 5.6 *For a given target concept* t(a, x) *and a related fact such as* p(a, b, num)*, where* a *and* b *are nominal values and* num *is a numeric value; instead of a single rule, the following two rule are generated:*

$$t(a,x) \leftarrow p(a,b,A), A \geq num.$$
$$t(a,x) \leftarrow p(a,b,A), A \leq num.$$

Once the comparison is defined on numeric attributes, aggregate predicates are included into C^2D. For this purpose, one-to-many relationships between target concept and background relations are defined according to schema information. For such relationships, aggregate predicates are generated by using pre-defined SQL commands. In the generalization step, the instances of these predicates are considered for concept rule generation.

For the *atom_count* predicate defined in Mutagenesis database, the aggregate value (second argument) has infinite domain. The first concept instance is *molecule(d1, true)* and atom_count(d1, 28) is a related fact. Due to this fact, the following candidate rules are created in the generalization step.

$$molecule(d1, true) \leftarrow atom\_count(d1, A), A \geq 28.$$
$$molecule(d1, true) \leftarrow atom\_count(d1, A), A \leq 28.$$

To test this modification in C^{2}D, a smaller mutagenesis data set was prepared. In this data set, *molecule* relation has 18 (9 true and 9 false), *atom* relation has 69, *bond* relation has 67 and both *atom_count* and *bond_count* relations have 18 records. As there are 18 different drugs, the *drug* (type table) relation has 18 records.

The *atom* relation has five arguments and the fifth argument, namely *charge*, has infinite domain. Similar to aggregate predicates, there will be two candidate rules, in which one rule has $\leq$ operator for constant values of *charge* argument and one rule has $\geq$ operator.

The minimum support threshold is defined as 0.2, minimum confidence as 0.6 and B as 1. In addition, recursion is disallowed in this test. The test results are shown in Table 5.8.

Table 5.8: The rules found on the smaller mutagenesis data set

Rule	**Support**	**Confidence**
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,B,C,D,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,B,C,F,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,B,F,D,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,B,F,G,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,F,C,D,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,F,C,G,E), E≤0.142	9/18	9/13
molecule(A,true) ← atom(A,B,C,D,E), E≥0.011, atom(A,F,G,D,E), E≤0.142	9/18	9/13
molecule(A,false) ← bond_count(A,B), B≤3	8/9	8/9

In other words, if a drug has an atom which has charge between 0.011 and 0.142 it is mutagenic. If a drug has less than (or equal to) 3 bonds, then it is non-mutagenic. These rules can not be find by C^{2}D without aggregate predicates. As seen above, the rules are very strong and meaningful (high support/confidence). This test shows the importance of aggregate predicates in ILP-based MRDM systems.

5.4 Constructing Transitive Rules in C^{2}D

In this section, formal definitions for inducing transitive rules in the proposed method are given below:

Definition 5.6 *For the target concept instance* $t(a_1, ..., a_n)$ *(a_i's are constants), the background facts which contain any* a_i *(with the same type according to* a_i *in target instance) are* related facts *of* $t(a_1, ..., a_n)$.

To state in a more informal way, any background fact sharing a constant argument with the target instance is a *related fact* of the target instance. As an example, for the target concept

instance *r(a,b)*, the background facts *m(a,c)* and *n(d,b,e)* are *related facts*.

Definition 5.7 *For the target concept instance* $t(a_1, ..., a_n)$ *(a_i's are constants), the background facts which do not contain any of the a_i's as arguments are called* unrelated facts *of* $t(a_1, ..., a_n)$.

In contrast to related facts, *unrelated facts* do not have any common arguments with the given target instance. For example, *n(e,g)* and *p(h)* are *unrelated facts* of *r(a,b)*.

Definition 5.8 *For the target concept instance* $t(a_1, ..., a_n)$ *(a_i's are constants), and the related fact* $f(b_1, ..., b_m)$*, the background facts which do not contain any a_i but contain at least one b_i (with same type as b_i in f) are* indirectly related facts *of* $t(a_1, ..., a_n)$.

Some of the unrelated facts may be an *indirectly related fact* for the given target instance. For example, given the target instance *t(a,b)* and related fact *m(a,c)*, the background instance *n(c,d)* is an *indirectly related fact* of *t(a,b)*.

On the basis of related, unrelated and indirectly related fact definitions, we can define related, unrelated and indirectly related relations.

Definition 5.9 *For the target concept instance* $t(a_1, ..., a_n)$ *(a_i's are constants), a background relation (predicate)* r *is a* related relation *to* t *if* r *has any instance that is a related instance to* $t(a_1, ..., a_n)$*. Similarly, for the target concept instance* $t(a_1, ..., a_n)$*, a background relation* r *is an* unrelated relation *(predicate) to* t *if* r *does not have any instance that is a related instance to* $t(a_1, ..., a_n)$*. By using the related and unrelated relation definitions, we can define* indirectly related relation *as follows: For the target concept instance* $t(a_1, ..., a_n)$*, if* r *is a related relation to* t, f *is an unrelated relation to* t, *and* f *is a related relation to some instance of* r *that is a related fact to* $t(a_1, ..., a_n)$*, then* f *is an indirectly related relation to* t.

Definition 5.10 *For the target concept predicate* $t(X_1, ..., X_n)$ *and the background predicates* $b_1(Y_1, ..., Y_m), ..., b_k(Z_1, ..., Z_r)$*, a* transitive rule *is a rule, in which*

- *target predicate is the head of the rule,*
- *some or all of the background predicates take part in the body of the rule,*
- *the variable arguments of the head predicate can appear only in the first predicate of the body, and*

- *the variable arguments of i^{th} body predicate can appear only in the $(i + 1)^{th}$ body predicate (except for the last body predicate which have common arguments only with the preceding predicate)*

For example, for the target predicate *t(A, B)* and the background predicates *f(C, D)* and *r(E, F, G)*, the following is a transitive rule.

t(X, Y) ← f(X, Z), r(Z, U, V). (X, Y, Z, U and V are variables)

In the earlier version, C^2D considers only related facts to construct the 2-predicate (one head and one body predicate) generalized rules. Due to inverse resolution in the generalization step, a related relation may take part in the rule body without having any common attribute with the head. Therefore, this earlier approach can generate transitive rules. However, this mechanism falls short for the domains including background predicates that are unrelated relations to all target facts. Then, they can never take part in rule generation. Michalski's trains problem [63] is a typical case for this situation. In this data set, the target relation *eastbound(train)* is only related with *has_car(train, car)* relation. The other background relations are only related with *has_car* relation.

Table 5.9: The relations in the train example

Relation Name	**Argument Types**
eastbound	train
has-car	train, car
short	car
closed	car
long	car
open-car	car
double	car
jagged	car
shape-car	car, shape
load	car, shape, number
wheel	car, number

The previous version of C^2D precedes as follows:

The *eastbound* relation has 5 records and the system takes the first target instance (*east1*). The target relation has one parameter and its type is *train*. Only (*has_car*) relation is related with *eastbound* and the other background relations are not related. So, it is not possible to join different relations in the specialization phase. Because of this, C^2D finds rules that include

only *has_car* relation in the body. This leads to the problem that generated rules are very general and do not reflect the characteristic of the *train* concept.

eastbound(A) ← has_car(B, car_11).

eastbound(A) ← has_car(B, car_12).

eastbound(A) ← has_car(B, car_13).

eastbound(A) ← has_car(B, car_14).

eastbound(A) ← has_car(east1, B).

eastbound(A) ← has_car(A, B).

eastbound(A) ← has_car(B, C).

As seen above, the generated rules are very general and can not include any information about the properties of the *cars* of the train.

In order to solve this problem, generalization step of C^2D, which is described in Section 5.2, is modified as follows:

(New) Generalization: Let $t(a_1,...,a_n)$ be an uncovered positive example. As the first step, set S that contains all related facts of $t(a_1,...,a_n)$, is generated. As the second step, set S' that contains related facts of each element of set S, is generated. Note that S' contains indirectly related facts of $t(a_1,...,a_n)$. Thirdly, set S is set to be $S \cup S'$. With the elements of set S, two literal generalizations of t are generated. The second and third steps constitute the modifications for including indirectly related relations.

In the Michalski's train example, the first uncovered target instance is *eastbound(east1)*. For this instance, the set S is generated as {has_car(east1, car_11), has_car(east1, car_12), has_car(east1, car_13), has_car(east1, car_14)}, the set S' is generated as {closed(car_12), ..., load(car_11, rectangle, 3)}. Therefore, set S is set to be {has_car(east1, car_11), ..., load(car_11, rectangle, 3)}. As a result of this extension, C^2D finds the following transitive rule:

For example, the first target instance is eastbound(east1) and it is only related with the following facts:

eastbound(A) ← has_car(A, B), closed(B). (s=5/5, c=5/7).

This extension to generalization phase is added as an optional property as *add indirectly related facts* into C^2D implementation. If this option is selected, the indirectly related facts of the target concept are added to the APRIORI lattice in the generalization step, otherwise only related facts are used.

CHAPTER 6

CONCEPT RULE INDUCTION SYSTEM (CRIS)

In the basic algorithm of C^2D, the experiments show that the selection order of the target instance (the order in the target relation) may change the result hypothesis set. In each coverage set, the induced rules depend on the selected target instance and the covered target instances in each step do not have any effect on the induced rules in the following coverage steps.

As a remedy to this problem, a new mechanism is developed and used in the improved version of C^2D, namely Concept Rule Induction System (CRIS) [43, 45].

6.1 The Algorithm of CRIS

In the algorithm of CRIS (the flowchart is shown in Figure 6.1), the generalization step of C^2D is modified to improve the rule quality.

As shown in the flowchart given in Figure 6.1, concept rule induction algorithm of CRIS takes target relation and background facts from the database. It works under minimum support, minimum confidence and maximum rule depth parameters. Rule construction starts with the calculation of feasible values for the head and body relations in order to generate most general rules with a head and a single body predicates. In generalization step, primary-foreign key relationship (Strategy 5.4) is also used in most general rule construction.

Following generalization step, the concept rule space is searched with an Apriori-based specialization operator. In this step, θ-subsumption (Strategy 5.1) is employed for candidate rule generation. In the refinement graph, infrequent rules are pruned. In addition to this, on the basis of Strategy 5.3, rules whose confidence values are not higher than that of their parents are also eliminated.

When the maximum rule depth is reached or no more candidate rules can be found, the rules that are below the confidence threshold are eliminated for the solution set. Among the

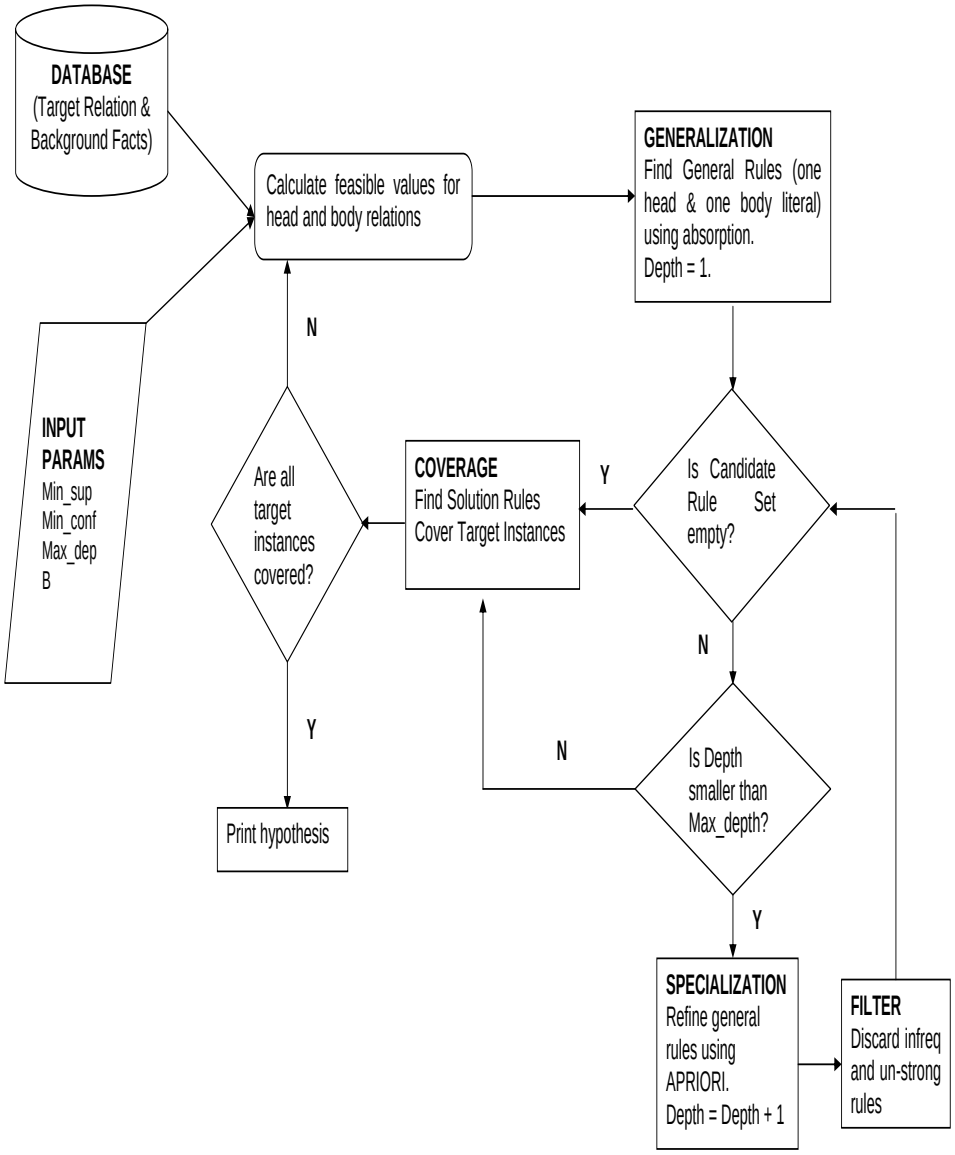

Figure 6.1: The flowchart in CRIS algorithm

produced strong and frequent rules, the best rule (with the highest f-metric value) is selected. The rule search is repeated for the remaining concept instances that are not in the coverage of the generated hypothesis rules. At the, some uncovered positive concept instances may exist because of the user settings for the thresholds. In the rest of this section, the main steps of the algorithm are described.

Generalization: Generalization step of the algorithm constructs the most general two-literal rules by considering all target instances together. By this way, the quality of the rule

induction does not depend on the order of target instances. This novel technique proceeds as follows.

For a given target relation such as *t(A, B)*, the induced rule has a head including either a constant or a variable for each argument of *t*. Each argument can be handled independently in order to find the feasible head relations for the hypothesis set. As an example, for the first argument *A*, a constant must appear at least $min\_sup * number\_of\_uncovered\_instances$ times in the target relation so that it can be used as a constant in an induced rule. In order to find the feasible constants for the attribute *A*, the SQL statement given in Table 6.1 is executed.

Table 6.1: The SQL query for finding feasible constants

```
SELECT a
FROM t
GROUP BY a
HAVING COUNT(*) ≥ (min_sup * num_of_uncov_inst)
```

For example, in the PTE-1 database [97] the target relation *pte_active* has only one argument (*drug*). Initially, there are 298 uncovered instances in *pte_active*. When the min_sup parameter is set as 0.05, the SQL statement given in Table 6.2 returns empty set which means there can not be a constant for the argument *drug* of *pte_active*. Therefore, the argument *drug* of *pte_active* can only be a variable for the head of the solution hypothesis rules.

Table 6.2: The SQL query example for support calculation

```
SELECT drug
FROM pte_active
GROUP BY drug
HAVING COUNT(*) ≥ (298 * 0.05)
```

In the same manner, for a background relation such as *r(a, b, c)*, if a constant appears at least $min\_sup * number\_of\_instances$ times for the same argument in *r*, then it is a frequent value for that argument of *r* and may take part in the solution rule for the hypothesis set. As an example, in the PTE-1 database, *pte_atm(drug, atom, element, integer, charge)* is a background relation and the feasible constants which can take part in the hypothesis set can be found for each argument of *pte_atm* by using the above SQL statement template.

For numeric attributes, due to support threshold, it is not feasible to seek for acceptable constants. For this reason, feasible ranges are given through *less-than/greater-than* operators on constants. As an example, for the *charge* argument of *pte_atm* predicate, all values in the database are sorted in ascending order. For min_sup 0.05, there should be 19 (which is (1/0.05)-1) border values for each less-than/greater-than operator. If pte_atm relation has 1000 records, after ordering from smallest to largest, less-than/greater-than operator is applied for the 51^{th} constant, 101^{th} constant and so on. In addition to these constants denoting feasible ranges, this argument can be a variable, as well.

The possible constants for each argument of *pte_atm* are as follows (The SQL statements for each argument are given in Appendix B):

drug: empty set (only variable)

atom: empty set (only variable)

element: c, h, o (also variable)

integer: 3, 10, 22 (also variable)

charge: less/greater operators applied on 19 constants (also variable)

As a result, *pte_atm* relation has 320 (which is, 1*1*4*4*20) body relations for each possible head relation in the generalization step of CRIS. Example generalized rules are listed in Table 6.3.

Table 6.3: Example generalized rules for PTE-1 data set

pte_active(A) ← pte_atm(A, B, c, 3, X), X ≤ -0.133.
pte_active(A) ← pte_atm(A, B, c, 3, X), X ≥ -0.133.
pte_active(A) ← pte_atm(A, B, c, 3, C)
pte_active(A) ← pte_atm(A, B, c, 10, X), X ≤ -0.133.
pte_active(A) ← pte_atm(A, B, c, 10, X), X ≥ -0.133.
pte_active(A) ← pte_atm(A, B, c, 10, C)
pte_active(A) ← pte_atm(A, B, c, 22, X), X ≤ -0.133.
pte_active(A) ← pte_atm(A, B, c, 22, X), X ≥ -0.133.
pte_active(A) ← pte_atm(A, B, c, 22, C)
pte_active(A) ← pte_atm(A, B, c, C, X), X ≤ -0.133.
pte_active(A) ← pte_atm(A, B, c, C, X), X ≥ -0.133.
pte_active(A) ← pte_atm(A, B, c, C, D)
pte_active(A) ← pte_atm(A, B, h, 3, X), X ≤ -0.133.
pte_active(A) ← pte_atm(A, B, h, 3, X), X ≥ -0.133.
pte_active(A) ← pte_atm(A, B, h, 3, C)

In the daughter example (Table 3.1), the target and background relations can only have

variables for the arguments in the hypothesis set. Under the support threshold value 0.8, among 13 rules generated for *daughter* database, only 6 rules (shown in bold) satisfy the threshold. The support values of two literal candidate rules are given in Table 6.4.

Table 6.4: The support values of two literal concept rules generated in CRIS

Concept Description	**Support Value**
daughter(A, B) ← parent(A, A)	0.0
daughter(A, B) ← parent(A, B)	0.0
daughter(A, B) ← parent(A, C)	0.0
daughter(A, B) ← parent(B, A)	**1.0**
daughter(A, B) ← parent(B, B)	0.0
daughter(A, B) ← parent(B, C)	**1.0**
daughter(A, B) ← parent(C, A)	**1.0**
daughter(A, B) ← parent(C, B)	0.5
daughter(A, B) ← parent(C, C)	0.0
daughter(A, B) ← parent(C, D)	**1.0**
daughter(A, B) ← female(A)	**1.0**
daughter(A, B) ← female(B)	0.5
daughter(A, B) ← female(C)	**1.0**

The specialization, evaluation and coverage steps are the same as in C^2D, which are described in Section 5.2.

At the end of the algorithm, the rule *daughter(A, B)* ← *parent(B, A), female(A)* with support value of 1.0 and the confidence value of 1.0 (f-metric=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 following hypothesis:

$$daughter(A, B) \leftarrow parent(B, A), female(A).$$

6.2 Aggregate Predicates in CRIS

In relational database queries, aggregate functions characterize groups of records gathered around a common property. In concept discovery, aggregate functions are utilized in order to construct aggregate predicates that capture some aggregate information over one-to-many relationships. In Section 5.3, aggregate predicates are formally defined for C^2D. In this section, aggregate predicates are created in a similar way to C^2D [45].

As an example, PTE-1 database [97] is used for explaining how aggregation is used in concept discovery in CRIS. There is one-to-many relationship between *pte_active* and *pte_atm* re-

lations over the *drug* argument. A similar relation exists between the *pte_active* and *pte_bond* tables. Also there is a one-to-many relationship between *pte_atm* and *pte_bond* relations over the *atm-id* argument.

$pte\_atm\_count^{atom;atm-id}_{drug;COUNT}(drug, cnt)$ is an example aggregate predicate that can be defined in the PTE-1 database. For simplicity, we abbreviate it as *pte_atm_count(drug, cnt)* which represents number of atoms for each drug. The instances of *pte_atm_count(drug,cnt)* aggregate predicate on *pte_atm* relation are constructed by the query given in Table 6.5.

Table 6.5: SQL statement for aggregate predicate pte_atm_count(drug, cnt)

```
SELECT drug, COUNT(atm-id) as cnt
FROM pte_atm
GROUP BY drug
```

All aggregate predicates defined on PTE-1 data set, their descriptions and the corresponding SQL query definitions are listed in Table 6.6.

Table 6.6: The aggregate predicates in PTE-1 data set

Predicate	**Description**	**SQL Query Definition**
pte_atm_count (drug, cnt)	Number of atoms for each drug	SELECT drug,COUNT(atm-id) FROM pte_atm GROUP BY drug
pte_bond_count (drug, cnt)	Number of bonds for each drug	SELECT drug,COUNT(atm-id) FROM pte_bond GROUP BY drug
pte_atm_b_cnt (atm-id, cnt)	Number of bonds for each atom	SELECT atm-id1,COUNT(atm-id2) FROM pte_bond GROUP BY atm-id1
pte_charge_max (drug, mx)	Max charge of the atoms in a drug	SELECT drug,MAX(charge) FROM pte_atm GROUP BY drug
pte_charge_min (drug, mn)	Min charge of the atoms in a drug	SELECT drug,MIN(charge) FROM pte_atm GROUP BY drug

Aggregate predicates have numeric attributes by their nature. Therefore, in order to add aggregate predicates into the system, numeric attribute types should also be handled. Since it is not useful and feasible to define concepts on specific numeric values, in this thesis, numeric

attributes are considered only together with comparison operators. For example, the *pte_atm* relation in the above example has the argument *charge* which has floating-point values. It is infeasible to search for a rule such as: *A drug is active if it has an atom with charge equals to -0.117.* As there are many possible numeric values in the relation, such a rule would probably be eliminated according to minimum support criteria. Instead, to search for drugs which has charge larger/smaller than some threshold value will be more feasible. For this purpose, numeric attributes are handled as described below.

As the first step, domain of the numerical attributes are explicitly defined as *infinite* in the generalization step. For the *infinite* attributes, concept rules are generated on the basis of Strategy 5.6 (given in Section 5.3).

In order to find the most descriptive values for numerical attributes, the basic method is ordering the domain values for the numeric attribute and defining the intervals with respect to the given support threshold. Therefore, a set of rules describing the interval borders are generated. This method is described in Section 6.1. It is also applicable for numeric attributes of the aggregate predicates, as well. However, the number of generalized rules highly increases under low support threshold. For this reason, in order to improve the time efficiency, a simplification is employed and only the median element of the domain is selected as the *num* value.

The integration of aggregate predicates into the concept rule generation process can be summarized as follows. One-to-many relationships between target concept and background relations are defined on the basis of the schema information. Under these relationships, aggregate predicates are generated by using the SQL template as described in earlier in this section. In the generalization step, the instances of these predicates are considered for rule generation [45].

As an example, for the *pte_atm_count* predicate defined in PTE-1 database, the following example rules are created in the generalization step.

$$pte\_active(A, true) \leftarrow pte\_atm\_count(A, X), X \geq 22.$$
$$pte\_active(A, true) \leftarrow pte\_atm\_count(A, X), X \leq 22.$$

6.3 Constructing Transitive Rules in CRIS

To induce transitive rules in the basic algorithm of C^2D, an option is implemented as described in Section 5.4. However, as the CRIS handles all the background relations independently, it can find the transitive rules in the search space without any special treatment.

For the eastbound train example given in Section 5.4, CRIS takes all of the background relations into consideration and finds the following rules after the generalization step.

Table 6.7: Concept rules induced after the generalization step of CRIS in the train example

Possible Rules
eastbound(A) ← has-car(A, B)
eastbound(A) ← has-car(B, C)
eastbound(A) ← short(B)
eastbound(A) ← closed(B)
eastbound(A) ← long(B)
eastbound(A) ← open-car(B)
eastbound(A) ← double(B)
eastbound(A) ← jagged(B)
eastbound(A) ← shape-car(B, C)
eastbound(A) ← load(B, C, D)
eastbound(A) ← wheel(B, C)

As a result, CRIS finds the following transitive rule in the search space:

eastbound(A) ← has_car(A, B), closed(B). (s=5/5, c=5/7).

This rule can be found in C^2D with extended generalization. CRIS finds the rule in half of the C^2D's solution time.

CHAPTER 7

EXPERIMENTAL RESULTS

A set of experiments were performed to test the algorithm of C^2D and CRIS on well-known problems in terms of coverage and predictive accuracy. Coverage denotes the number of target instances of test data set covered by the induced hypothesis set. Predictive accuracy denotes the sum of correctly covered true positive and true negative instances over the sum of true postive, true negative, false positive and false negative instances. [1] The experiments were run on a notebook computer having Intel Core Duo 1.6 Ghz processor and 1 GB memory.

7.1 Linear Recursive Rule Learning

One of the interesting test cases that we have used is a complex family relation, "same-generation" learning problem. In this experiment, only linear recursion is allowed and B value is set to be 1. We set the confidence threshold as 0.6, support threshold as 0.3 and maximum depth as 3.

In the data set, 344 pairs of actual family members are given as positive examples of *same-generation (sg)* relation. Additionally, 64 background facts are provided to describe the *parental (p)* relationships in the family. The tables *sg* and *p* have two arguments having *person*. As there are 47 persons in the examples, the *person* table (type table) has 47 records.

The solutions under different evaluation criteria are given in Table 7.1 (the parameters in lower-case letters are constants that exist in the data set). The row titles *Confidence* and *F-Metric* denote confidence criterion alone and f-metric evaluation, respectively.

As seen from the results in Table 7.1, improved confidence evaluation can find better rules than the conventional confidence evaluation. Among the improved criteria, f-metric produce

[1] In order to find the false positive and false negative instances, the test data set is extended with the dual of data set under CWA.

Table 7.1: Rules found on the same generation data set

Conventional	Confidence	sg(A, B) ← p(C, A).
		sg(A, B) ← p(C, B).
	F-Metric	sg(A, B) ← p(C, A).
		sg(A, B) ← p(C, B).
Improved	Confidence	sg(A, B) ← sg(C, D), p(C, A), p(D, B).
		sg(A, B) ← sg(A, neriman), p(yusuf, B).
		sg(A, B) ← sg(B, ali), p(mediha, A).
		sg(A, B) ← p(yusuf, A), p(yusuf, B).
		sg(A, B) ← p(mediha, A), p(mediha, B)
		sg(A, B) ← p(C, A), p(C, B).
	F-Metric	sg(A, B) ← sg(C, D), p(C, A), p(D, B).
		sg(A, B) ← sg(C, D), p(C, B), p(D, A).
		sg(A, B) ← p(C, A), p(C, B).

better rules than using only confidence for hypothesis evaluation.

The first two concept rules of the solution using the settings f-metric evaluation with improved confidence show that "same-generation" relation is a symmetric relation and the third rule forms the base rule for the recursive solution.

CRIS also finds the correct hypothesis set for this data set in a shorter time (half time with respect to C^2D).

For this data set, ALEPH, PROGOL and GOLEM can not find a solution under default settings. Under strong mode declarations and constraints, ALEPH finds the following hypothesis:

sg(A, B) ← p(C, A), p(C, B).
sg(A, B) ← sg(A, C), sg(C, B).
sg(A, B) ← p(C, A), sg(C, D), p(D, B).

However, PROGOL can only find "sg(A, B) ← sg(B, C), sg(C ,A)." as a solution. Similarly, GOLEM could not find any solution under strong mode declarations.

7.2 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 deformation 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 geometric position.

Four different structures called (*b*-*e*) in [23] are used for learning in this experiment. The structure *a* is used for testing the accuracy and coverage of the induced concept rules. 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 examples *(c15, 8)*, means that edge 15 of the structure *c* should be divided in 8 sub-edges. The name and arguments of the relations in the data set are given in Table 7.2.

Table 7.2: The relations in the mesh-design data set

Relation Name	**Argument Types**
circuit	mesh
circuit_hole	mesh
cont_loaded	mesh
element (type reln.)	mesh
equal	mesh, mesh
fixed	mesh
free	mesh
half_circuit	mesh
half_circuit_hole	mesh
long	mesh
long_for_hole	mesh
mesh_train (target reln.)	mesh, NumberOfEdges
mesh_test (test reln.)	mesh, NumberOfEdges
neighbour_xy	mesh, mesh
neighbour_yz	mesh, mesh
neighbour_zx	mesh, mesh
nnumber (type reln.)	integer
noload	mesh
not_important	mesh
one_side_fixed	mesh
one_side_loaded	mesh
opposite	mesh, mesh
quarter_circuit	mesh
short_for_hole	mesh
short	mesh
two_side_fixed	mesh
two_side_loaded	mesh

There are 223 positive training examples and 1474 background facts in the data set. The target relation *mesh_train* has two arguments having *element* and *integer* type. The type tables *element* and *integer* are created having 278 and 13 records. The test relation *mesh_test* has 55 examples.

For this experiment, recursion is disallowed, support threshold is set as 0.1, B is set as 1 and maximum depth is set as 3. We test the data set on several confidence thresholds (0.1 through 0.5). The details of the results and coverage of previous systems are shown in Table 7.3.

Table 7.3: Test results for the mesh-design data set

System		**Coverage (over 55 records)**	**Pred. Acc.**
FOIL		17	
GOLEM		17	
PROGOL		17	
MFOIL		19	
SAHILP		21	
PosILP		23	
ALEPH (strict decl.)		26	
C^2D (with min-conf)	**0.1**	**31**	**0.29**
	0.2	25	
	0.3	15	
	0.4	19	
	0.5	17	
CRIS		**29**	**0.49**

Selection of parameters is important to induce meaningful results for sparse data sets such as Mesh Design data set. For example, we get different results according to different minimum confidence threshold values. For some confidence thresholds, C^2D finds better results according to previous systems.

In another experiment, minimum confidence threshold is set as 0.1. As seen in Table 7.3, C^2D finds rules that cover 31 of the 55 records in the test data set. CRIS finds rules that cover 29 of the records in the test data set. However, the accuracy of the rules found by C^2D is 0.29, whereas the accuracy of the rules by CRIS is 0.49. In other words, the improved version finds rules that have nearly same coverage but higher accuracy according to the basic version of C^2D.

7.3 Predictive Toxicology Evaluation

A large percentage of cancer incidents stems from the environmental factors, such as cancerogenic compounds. The carcinogenecity 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 [21].

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 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 [97]. The name and arguments of the relations in the PTE data set are given in Table 7.4.

The background knowledge has roughly 25,500 facts [96]. The target relation *pte_active* has two arguments having *drug* and *bool* type. The primary key for the target relation is *drug* and it exists in all background relations as a foreign key. The type tables *drug* and *bool* are created having 340 and 2 (true/false) records respectively.

For this experiment, recursion is disallowed, support threshold is set as 0.05, confidence threshold is set as 0.7, B is set as 1 and maximum depth is set as 3. The predictive accuracy of the hypothesis set is computed by the proportion of the sum of the carcinogenic concept instances classified as positive and non-carcinogenic instances classified as negative to the total number of concept instances that the hypothesis set classifies.

There is one-to-many relationship between *pte_active* and *pte_atm* relations over the *drug* argument. Similar relation exists between the *pte_active* and *pte_bond* tables. By using these relationships, the following aggregate predicates are added to the background knowledge; where *drug* is the key and *cnt* is the aggregate value for the aggregate predicate:

pte_atm_count(drug, cnt).

pte_bond_count(drug, cnt).

pte_atm_bond_count(atom, cnt).

pte_atm_charge_max(drug, mx).

pte_atm_charge_min(drug, mn).

Table 7.4: The relations in the PTE-1 data set

Relation Name	**Argument Types**
alcohol	drug, ring
alkyl-halide	drug, ring
ames	drug
amine	drug, ring
pte_atm	drug, atom, element, atom-type, charge
pte_bond	drug, atom, atom, bond-type
bool (type reln.)	boolean
drug (type reln.)	drug
ester	drug, ring
ether	drug, ring
five-ring	drug, ring
has-property	drug, property, value
imine	drug, ring
ind	drug, ind, value
ketone	drug, ring
methoxy	drug, ring
methyl	drug, ring
mutagenic	drug
nitro	drug, ring
non-ar-5c-ring	drug, ring
non-ar-6c-ring	drug, ring
non-ar-hetero-5-ring	drug, ring
non-ar-hetero-6-ring	drug, ring
phenol	drug, ring
pte-train (train reln.)	drug, boolean
pte-test (test reln.)	drug, boolean
six-ring	drug, ring
sulfide	drug, ring
sulfo	drug, ring

The predictive accuracies of the state-of-art methods and C^2D (with aggregation) for PTE-1 data set are listed in Table 7.5. As seen from the table, only Ashby has a better performance than C^2D. However, Ashby is a special system which is developed for this kind of problem sets. The reader may refer to [96] for more information on the compared systems in Table 7.5.

In another experiment, min_sup threshold is set as 0.1. For PTE-1 data set, the aggregate predicates given in Section 6.2 are defined and their instances are added to the background information. For numeric attributes, less than/greater than operators are applied for calculated interval boundary values. But, this makes the search space very large and the experiments do

Table 7.5: Predictive accuracies of C^2D and similar systems for the first experiment on PTE-1 data set

Method	**Type**	**Predictive Accuracy**
Ashby	Chemist	0.77
C^2D with aggregation	**ILP + DM**	**0.75**
PROGOL	ILP	0.72
RASH	Biological Potency Analysis	0.72
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

not finish in feasible time. Due to this, only the median element in the ordered sequence is taken as an acceptable constant for which less than/greater than operators will be applied. As an example, for the fifth argument of *pte_atm* relation (charge), first of all, the corresponding values in the database are ordered in the ascending order. The *pte_atm* relation has 9189 records, then after ordering from smallest to largest, less than/greater than operator is applied for the 4595^th constant. In addition, this argument can be a variable, as well.

The predictive accuracies of the state-of-art methods, C^2D and CRIS for PTE-1 data set are listed in Table 7.6. As seen from the table, CRIS has a better predictive accuracy than the basic C^2D algorithm. In addition, it finds the best results (having highest accuracy) with respect to other systems.

An example rule including an aggregate predicate is shown below:

pte_active(A, false) ← pte_atm(A, B, c, 22, X), X >= -0.020, pte_has_property(A, salmonella, n), pte_has_property(A, mouse_lymph, p).

Within this experiment, the effect of including aggregate predicates in execution time of the system is experimentally analyzed. For this experiment, CRIS runs on PTE-1 data set is with none, one and five aggregate predicates included in the background knowledge. The result is presented in Figure 7.1. As seen in the figure, including a single aggregate predicate in rule discovery mechanism causes a high increase in execution time (i.e., duration of concept discovery). However, the increase rate drops in the inclusion of new aggregate predicates. For the domains, where the aggregate predicates are descriptive for the concept, experimentally observed increase rate in execution time can be tolerated. Furthermore, the

Table 7.6: Predictive accuracies of CRIS and C^{2}D for the second experiment on PTE-1 data set

Method	Type	Pred. Acc.
CRIS (with aggr.)	**ILP + DM**	**0.88**
CRIS	**ILP + DM**	**0.86**
Ashby	Chemist	0.77
PROGOL	ILP	0.72
RASH	Biological Potency An.	0.72
C^{2}D (with aggr.)	**ILP + DM**	**0.70**
TIPT	Propositional ML	0.67
Bakale	Chemical Reactivity An.	0.63
Benigni	Expert-guided Regr.	0.62
DEREK	Expert System	0.57
TOPCAT	Statistical Disc.	0.54
COMPACT	Molecular Modeling	0.54

number of aggregated predicates included in the system can be more than one, since the cost of adding more than one aggregate predicate is not much higher than including a single aggregation predicate.

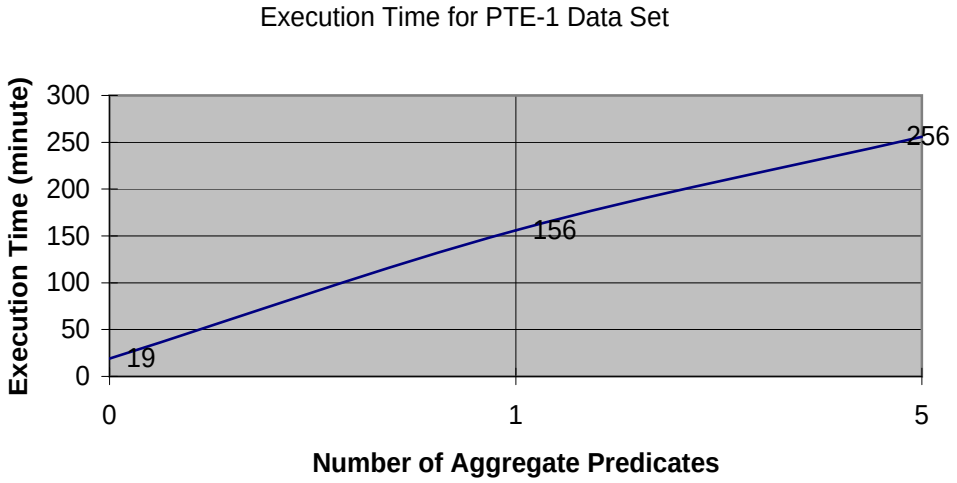

Figure 7.1: Execution time for concept discovery with aggregation in PTE-1 data set

7.4 Mutagenicity Test

In the fourth experiment, we have studied the mutagenicity of 230 compounds listed in [98]. We use the *regression-friendly* data set which has 188 compounds. The target relation *molecule* has two arguments having *drug* and *bool* type. The primary key for the target relation is *drug* and it exists in all background relations as a foreign key. The type tables *drug* and *bool* are created having 230 and 2 (true/false) records respectively. The structure of the target table and two important background relations in the database are shown in Figure 4.1.

In the literature [94] five levels of background knowledge are given for Mutagenesis as described in Section 4.1. Five sets of background knowledge are defined in the data set where $B_i \subset B_{i+1}$ for $i = 0..3$. In this experiment, B_2 is used.

In this experiment, recursion is disallowed, support threshold is set as 0.1, confidence threshold as 0.7, B is set as 1 and maximum depth is set as 3.

The following aggregate predicates, which were described in Section 5.3 are created and instances are added to background knowledge.

atom_count(drug, cnt).

bond_count(drug, cnt).

For description of the target concept (*molecule*), C^{2}D finds the following aggregate rule:

molecule(A, true) ← atom(A, B, C, 29, D), bond_count(A, E), E >= 27.

The predictive accuracies of the state-of-art methods and C^{2}D for Mutagenesis database are listed in Table 7.7 [59].

Table 7.7: Predictive accuracies for the mutagenesis data set

Method	**Predictive Accuracy**
CRIS without aggregation	**0.95**
PosILP	0.90
SAHILP	0.89
MRDTL	0.88
C^{2}D with aggregation	**0.85**
TILDE	0.85
PROGOL	0.83
FOIL	0.83

As seen from the results, CRIS has the highest accuracy in this experiment. Although MRDTL has higher accuracy than C^2D, C^2D has a high coverage as 0.53 (for the best rule), i.e. it finds rules which have both high accuracy and coverage. CRIS finds the same best rule as in C^2D. For the best rule, SG (the technique in MRDTL) has a coverage as 0.28 and GSG (adding aggregation) has a coverage as 0.36 [48]. C^2D finds a set of rules which totally cover 164 of the 188 records. These coverage results (for the best rule) are listed in Table 7.8.

Table 7.8: Coverage values for the best rule in mutagenesis data set

Method	**Coverage (over 188 records)**	**Coverage (%)**
C^2D with aggregation	**99**	**53**
SG (MRDTL)	53	28
GSG	68	36

7.5 Constructing Transitive Rules Under Unrelated Facts

The original generation phase of C^2D considers only related facts to construct the 2-predicate (one head and one body predicate) generalized rules. However, this approach falls short for the cases where the domain includes many unrelated facts. Michalski's trains problem [63], which is shown in Table 5.9, is a typical case for this situation. In this data set, the target relation *eastbound(train)* is only related with *has_car(train, car)* relation. The other background relations have an argument of type *car* and are only related with *has_car* relation.

The previous version of generalization phase of C^2D precedes as follows:

The *eastbound* relation has 5 records which are {east1, east2, east3, east4, east5}. The system takes the first target instance which is *eastbound(east1)*. The target relation has one parameter and its type is train. One of the background relations (*has_car*) has only related column type and facts. The other background relations are not related. Therefore, it is not possible to join different relations in the specialization phase. As the result, C^2D finds the following rules for the *train* data set under min_sup 0.2 and min_conf 0.6.

eastbound(A) ← has_car(B, car_11).

eastbound(A) ← has_car(B, car_12).

eastbound(A) ← has_car(B, car_13).

eastbound(A) ← has_car(B, car_14).

eastbound(A) ← has_car(east1, B).

eastbound(A) ← has_car(A, B).

eastbound(A) ← has_car(B, C).

For this data set, the generated rules by C^2D are very general and can not include any information about the properties of the *cars* of the train. In C^2D, this problem is fixed by adding the background facts that are indirectly related with the selected target concept instance into APRIORI lattice in the generalization step. For example, the first target instance is *eastbound(east1)* and it is only related with the following facts:

has_car(east1, car_11).

has_car(east1, car_12).

has_car(east1, car_13).

has_car(east1, car_14).

The background facts which are related facts of the above facts, such as closed(car_12), load(car_11, rectangle, 3), are added as body of the fact rules in the generalization step. As a result of this extension, C^2D finds the following rule:

eastbound(A) ← has_car(A, B), closed(B). (s=5/5,c=5/7).

This extension to generalization phase is added as an optional property as *add indirectly related facts* into C^2D implementation. If this option is selected, the indirectly related facts of the target concept are added to the APRIORI lattice in the generalization step, otherwise only related facts are used.

As CRIS considers all target instances at once, it can find the same rule generated by C^2D in shorter time.

GOLEM can not find a rule for this experiment, however PROGOL finds only the following rule:

eastbound(A) ← has_car(A, B), double(B). (s=2/5,c=2/3).

ALEPH can not find a rule without negative instances. When negative instances are provided, it finds the following rule (best rule) for this experiment:

eastbound(A) ← has_car(A, B), short(B), closed(B). (s=5/5,c=5/5).

Another example for using indirectly related facts for transitive rule construction is the *kinship* data set that is adopted from [38]. The name and arguments of the relations in the data set are given in Table 7.9.

Table 7.9: The relations in the kinship data set

Relation Name	**Argument Types**
aunt	person, person
brother	person, person
daughter	person, person
father	person, person
husband	person, person
mother	person, person
nephew	person, person
niece	person, person
sister	person, person
son	person, person
uncle	person, person
wife	person, person

There are totally 271 records in the data set. As there are 24 different people in the relations, a *person* table (type table) is created which has the names of 24 people. In this experiment, a new relation called *elti(A,B)* was defined, which represents the family relation between the wives of two brothers. (The term elti is the Turkish word for this family relationship). In the data set, the people in *elti* relation have no brothers. Therefore, *brother* instances are unrelated facts of *elti*. The minimum support is set as 0.2 and minimum confidence is set as 0.6.

If *add indirectly related facts* option is not selected, then C^2D can not find high quality and semantically correct rules for the target relation *elti*. If this option is selected, then C^2D adds some records of the *brother* relation into the APRIORI lattice in the generalization step. Finally, it finds the following rules that can capture the description of *elti*:

elti(A, B) ← husband(C, A), husband(D, B), brother(C, D).

elti(A, B) ← husband(C, A), husband(D, B), brother(D, C).

elti(A, B) ← husband(C, A), wife(B, D), brother(C, D).

elti(A, B) ← husband(C, A), wife(B, D), brother(D, C).

elti(A, B) ← husband(C, B), wife(A, D), brother(C, D).

elti(A, B) ← husband(C, B), wife(A, D), brother(D, C).

elti(A, B) ← wife(A, C), wife(B, D), brother(C, D).

elti(A, B) ← wife(A, C), wife(B, D), brother(D, C).

For the same data set, GOLEM can not find a rule under several mode declarations. PROGOL can not find a meaningful rule for this experiment, as well. However, if only husband, wife and brother relations are given as background knowledge, then it finds only one of the transitive rule (given below) under strict mode declarations:

elti(A, B) ← husband(C, A), husband(D, B), brother(C, D).

ALEPH can only find one of the transitive rules for this experiment:

elti(A, B) ← husband(D, A), wife(B, C), brother(C, D).

CRIS also finds the correct hypothesis set for the above experiments.

The general overview of the experimental results are given in Table 7.10.

Table 7.10: The experimental results for train and elti data sets

Experiment	**C^2D Incl. Unrel. Facts**	**C^2D Without Unrel. Facts**	**CRIS**
Eastbound Train			
Accuracy	0.7	0	0.7
Coverage	1.0	0	1.0
Time (second)	8	1	5
Elti			
Accuracy	1.0	0.5	1.0
Coverage	1.0	0.5	1.0
Time (minute)	110	25	2.5

In order to test the scalability of CRIS for this experiment, a syntactic data set on *elti* experiment was prepared which has 2170 records (10 fictitious record for each record of each table in the original elti data set). CRIS can still find the same hypothesis with linear increase in time.

7.6 Constructing Transitive Rules Under Missing Background Information

Another observation about the transitive rule is that transitive rules may be constructed even though there is no unrelated facts in the domain. The related facts used in 2-predicate generalized rules can be combined in the specification phase in such a phase that the variable names are unified to form a transitive rule. In the other direction, we can say that under the proposed extension, even though the background has missing information, it is still possible to discover the rules that define the target concept.

As an example to such a situation, consider the *kinship* data set again. We extend this data set with a new relation called *dunur(A,B)* to represent the family relationship of two persons who are the parents of a married couple. (The term *dunur* is the Turkish word for this family relationship). The *dunur* relation has 16 records and has two arguments having *person* type. For example, *dunur(Penelope, Christine)* exists (and is true) in the relation because their children *Victoria* and *James* are married.

In the data set, the *dunur* relation is selected as the target relation, B is assigned as 1, minimum support as 0.2, minimum confidence as 0.6 and recursion is not allowed. In this run, *add indirectly related facts* option is not selected, which means that the indirectly related facts are not included in the generalization phase and the following rules are generated:

dunur(A, B) ← daughter(C, A), husband(D, C), son(D, B).
dunur(A, B) ← daughter(C, B), husband(D, C), son(D, A).
dunur(A, B) ← daughter(C, A), wife(D, C), son(D, B).
dunur(A, B) ← daughter(C, B), wife(D, C), son(D, A).

As seen above, the resulting rule set contains the transitive rules that captures the semantics of *dunur* predicate.

If some of the records, such as the husband or wife of the parents, are deleted in the data set, then *husband* and *wife* relations are not directly related to the *dunur* relation. (Note that it is a different situation than *elti* example. In this case, due to missing information, some of the related relations appear to be unrelated). When *add indirectly related facts* option is off again, the system can not generate successful rules as a solution. However, when *add indirectly related facts* option is selected, the above rules can be generated under the missing and reduced background information.

The same experiments are conducted on PROGOL, ALEPH and GOLEM systems. Neither of the systems could find any rules in both of the experiments either under strict mode

declarations.

CRIS finds the correct hypothesis set in both cases for this data set, as well.

The overview of results of the experiment are given in Table 7.11.

Table 7.11: The experimental results for *dunur* data set

Dunur Exp.	**C^{2}D Incl. Unrel. Facts**	**C^{2}D w/o Unrel. Facts**	**CRIS**
Accuracy	1.0	1.0	1.0
Coverage	1.0	0.75	1.0
Time (minute)	9	2	2.5

CHAPTER 8

CONCLUSION

ILP has become popular in computer science due to increase in the use of relational data and due to formalism provided by logic. Various systems are developed having different characteristics (search direction, language bias, etc). Each system has advantages and disadvantages depending on the input data.

In ILP systems, if there is not any language bias, it is impossible to define the target relation in complex and huge training data sets since search space grows very much. Therefore, one of the main tasks is to define the bias. However, by defining very strong bias, it is probable to miss the meaningful rules in the search process. As a result, there is trade-off between the correctness and efficiency in the algorithms. Another dimension of defining bias is dependence or independence of the bias declaration from the user. Mode declarations are important tools, since they separate bias definitions from the machinery and make it portable and adaptive specific to the problem. However, an important drawback of having mode declaration is that it is not straightforward to find the proper mode declaration for the problem by the user.

In real life, the relational data exists on databases and the data has only type definitions. For this reason, for an ILP system, the ability to work with the data stored in database is an important advantage to facilitate the concept discovery process. Integration with database also necessitate that the systems should not need negative instances as inputs, since there is databases consist of positive data.

In this thesis, an overview of ILP-based concept discovery systems is given at the beginning. Then, the well-known systems, LINUS, GOLEM, CIGOL, MIS, FOIL, PROGOL, ALEPH, WARMR and SAHILP are described and the basics of their concept discovery mechanisms are demonstrated on a running example. In addition to this, this thesis presents two ILP-based concept discovery systems, namely C^2D and CRIS. Both systems combine rule extraction methods in ILP and APRIORI-based specialization operator. By this way, strong

declarative biases are relaxed, instead, support and confidence values are used for pruning the search space. In addition, they do not require user specification of input/output modes of arguments of predicates and negative concept instances. Thus, they provide a suitable data mining framework for non-expert users who are not expected to know much about the semantic details of large relations, which are stored in classical database management systems.

Both C^2D and CRIS have a new confidence-based hypothesis evaluation criterion and confidence-based search space pruning mechanism. Conventional definition of the confidence, that is developed for query extension, is slightly modified by adding type predicates to the body of the query extension for the arguments of the head predicate that do not appear in the body. By this way, the domains are also included in the confidence calculation of the generated rules in concept discovery.

Confidence-based pruning is used in the candidate filtering phase. If the confidence value of the generated concept rule is not higher than confidence values of its parents, it means that, the specifications through it will not improve the hypothesis to be more confident. By this way, such rules are directly eliminated at early steps.

In some cases describing concepts using only background predicates may not be possible or may be very difficult. Recently, new concept discovery systems started to investigate other alternative ways to extend the rules for concept description. In order to generate successfull rules for the domains where aggregated values such as *sum, count* are descriptive in the semantics of the target concept, it is essential for a concept discovery system to support definition of aggregation and inclusion in the concept discovery mechanisms. In both C^2D and CRIS, aggregation information is defined in the form of aggregate predicates and they are included in the background knowledge of the concept. This leads to increase in execution time, however the concept discovery accuracy increases considerably. Due to the satisfactory results in rule quality, the decrease in time efficiency may be considered tolerable.

To be able to include aggregate predicates in the background knowledge, one-to-many relationships in the schema are given and aggregate predicates on such relations are activated through predefined aggregate queries. Due to space limitations, in this thesis, we have only presented the results for two experiments. In these experiments, we considered the popular aggregate functions such as *COUNT*, *MAX* and *MIN* as they are the relevant functions for the data sets. The implementation of other aggregate functions is also straightforward.

In CRIS, generalization step of C^2D is modified in such a way that most general rules are constructed by considering the number of occurrences of constant arguments in the rules. By

this mechanism, the effect of target instance order on rule generation is eliminated and rule quality is improved.

In this thesis, we also present a technique to generate rules that capture transitive relations. This is done by including unrelated facts in rule definitions in the first step (the most general rule generation step) of the concept discovery process. By this way, it is possible to generate rules that capture transitive relations through unrelated facts and to extract transitive rules under missing background information. Such rules are not very common, however, they become of importance for the cases where they are the only rules that describe the target concept. Another contribution of this technique is the ability to induce rules under missing background information.

Similar relational knowledge discovery systems can generate transitive rules as well. However, they use mode declarations for the attributes, which requires high level logic programming and domain knowledge. Under the proposed enhancements, C^2D and CRIS handle transitive rule generation without mode declarations. Inclusion of the unrelated facts extends the search space. However, experimentally, it is observed that this extension has a very little effect on the execution time. This feature of the system is also optional for C^2D, and it can be turned on only for the applications containing domains that have unrelated facts in the background information.

The proposed techniques are evaluated on several benchmark problems including same-generation, mesh design, predictive toxicoloy evaluation, mutagenicity and kinship tests. The experiments reveal promising test results that are comparable with the performance of current state-of-the-art knowledge discovery systems. Another important result deduced from the experiments is that improved confidence evaluation produces better concept descriptions than conventional confidence evaluation. For best rule selection, using f-metric leads to hypothesis with higher quality. Finally, in sparse data sets, it is important to select the parameters (such as min-conf threshold) to induce good results.

The experiments show that aggregate rules and comparison on numerical data provide better accuracy and coverage for data that has one-to-many relationships between the target and background relations. Although inclusion of aggregate predicates slightly drops the run-time efficiency, the increase in time can be considered negligible with respect to the increase in the accuracy of concept descriptions.

As a future work, both C^2D and CRIS can be integrated into data warehouses to induce association rules among multiple relations that exist in it.

REFERENCES

[1] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[2] 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.

[3] 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.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of 20^{t}h International Conference of Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, December 1994.

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

[6] Anneleen Assche, Celine Vens, Hendrik Blockeel, and Sašo Džeroski. First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64(1-3):149–182, 2006.

[7] A. Atramentov. Multi-relational decision tree algorithm - implementation and experiments. Master's thesis, Iowa State University, Iowa, USA, 2003.

[8] H. Blockeel. *Top-down Induction of First-order Logical Decision Trees*. PhD thesis, Katholike University, Leuven, Belgium, December 1998.

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

[11] 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.

[12] Ya-Wen Chang Chien and Yen-Liang Chen. A phenotypic genetic algorithm for inductive logic programming. *Expert Systems with Applications*, 36(3):6935–6944, 2009.

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

[14] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4), 1989.

[15] 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.

[16] A.K. Debnath, R.L. Lopez de Compadre, G. Debnath, A.J. Schusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Medicinal Chemistry*, 34(2), feb 1991.

[17] L. Dehaspe. *Frequent Pattern Discovery in First-Order Logic*. PhD thesis, Katholike University, Leuven, Belgium, December 1998.

[18] 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.

[19] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[20] 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.

[21] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 30–36. AAAI Press., 1998.

[22] L. Deraedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[23] B. Dolsak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

[24] P. Domingos. Prospects and challenges for multi-relational data mining. *SIGKDD Explorations*, 5:80–81, 2003.

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

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

[27] Saso Dzeroski. From inductive logic programming to relational data mining.

[28] Saso Dzeroski. *Relational Data Mining*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

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

[30] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communication of ACM*, 39(11):27–34, 1996.

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

[32] Richard Frank, Flavia Moser, and Martin Ester. A method for multi-relational classification using single and multi-feature aggregation functions. In *Principles of Data Mining and Knowledge Discovery*, pages 430–437, 2007.

[33] Lise Getoor and John Grant. Prl: A probabilistic relational language. *Machine Learning*, 62(1-2):7–31, 2006.

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

[35] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Colloquium on IR Research (ECIR'05)*, pages 345–359, Xerox Research Centre Europe 6, chemin de Maupertuis F-38240 Meylan, France, 2005. Springer.

[36] A. Atramentov H. Leiva and V. Honavar. Experiments with MRDTL - a multi-relational decision tree learning algorithm. In *Workshop on Multi-Relational Data Mining*, 2002.

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

[38] Geoff Hinton. UCI machine learning repository kinship data set, 1990. http://archive.ics.uci.edu/ml/datasets/Kinship.

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

[40] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Aggregation in confidence-based concept discovery for multi-relational data mining. In *Proceedings of IADIS European Conference on Data Mining (ECDM)*, pages 43–50, Amsterdam, Netherland, July 2008.

[41] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Confidence-based concept discovery in multi-relational data mining. In *Proceedings of International Conference on Data Mining and Applications (ICDMA)*, pages 446–451, Hong Kong, March 2008.

[42] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Analyzing transitive rules on a hybrid concept discovery system. In *LNCS, Hybrid Artificial Intelligent Systems*, volume 5572/2009, pages 227–234. Springer Berlin/Heidelberg, 2009.

[43] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Confidence-based concept discovery in relational databases. In *Proceedings of 2009 World Congress on Computer Science and Information Engineering (CSIE 2009)*, Los Angeles, USA, April 2009.

[44] Y. Kavurucu, P. Senkul, and I. H. Toroslu. ILP-based concept discovery in multi-relational data mining. *Expert Systems with Applications*, 36(9):11418–11428, November 2009.

[45] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Multi-relational concept discovery with aggregation. In *Proceedings of 24th International Symposium on Computer and Information Sciences (ISCIS 2009)*, Northern Cyprus, September 2009.

[46] 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.

[47] 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.

[48] A. J. Knobbe. *Multi-Relational Data Mining*. PhD thesis, Utrecht University, Utrecht, Netherlands, November 2004.

[49] Arno J. Knobbe, Arno Siebes, and Bart Marseille. Involving aggregate functions in multi-relational search. In *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 287–298, London, UK, 2002. Springer-Verlag.

[50] 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.

[51] R. A. Olshen L. Breiman, J. H. Friedman and C. J. Stone. Classification and regression trees, 1984.

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

[53] Evelina Lamma, Paola Mello, Michela Milano, and Fabrizio Riguzzi. Integrating induction and abduction in logic programming. *Inf. Sci. Inf. Comput. Sci.*, 116(1):25–54, 1999.

[54] Nada Lavrac, Peter A. Flach, and Blaz Zupan. Rule evaluation measures: A unifying view. In *ILP '99: Proceedings of the 9th International Workshop on Inductive Logic Programming*, pages 174–185, London, UK, 1999. Springer-Verlag.

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

[56] 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.

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

[58] Chien-I Lee, Cheng-Jung Tsai, Tong-Qin Wu, and Wei-Pang Yang. An approach to mining the multi-relational imbalanced database. *Expert Syst. Appl.*, 34(4):3021–3032, 2008.

[59] H. A. Leiva. MRDTL: A multi-relational decision tree learning algorithm. Master's thesis, Iowa State University, Iowa, USA, 2002.

[60] M. Liakata. *Inducing Domain Theories*. PhD thesis, University of Oxford, 2004.

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

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

[63] R. Michalski and J. Larson. Inductive inference of VL decision rules. In *Workshop on Pattern-Directed Inference Systems*, volume 63, pages 33–44, Hawaii, 1997. SIGART Newsletter, ACM.

[64] 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.

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

[66] 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.

[67] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

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

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

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

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

[72] 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.

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

[74] 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.

[75] Stephen Muggleton and Alireza Tamaddoni-Nezhad. QG/GA: A stochastic search for progol. *Machine Learning*, 70(2-3):121–133, 2008.

[76] Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, pages 625–630, New York, NY, USA, 2003. ACM.

[77] Swiss Federal Institute of Technology. Java graph visualization and layout.

[78] M. Pazzani and D. Kibler. The role of prior knowledge in inductive learning. *Machine Learning*, 9.

[79] Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, pages 167–176, New York, NY, USA, 2003. ACM.

[80] Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, pages 167–176, New York, NY, USA, 2003. ACM.

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

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

[83] J. R. Quinlan. C4.5: Programs for machine learning. San Mateo, CA, 1993. Morgan Kaufmann.

[84] 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.

[85] Jr. Roberto J. Bayardo, Rakesh Agrawal, and Dimitrios Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2-3):217–240, 2000.

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

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

[88] 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.

[89] 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.

[90] Mathieu Serrurier and Henri Prade. Introducing possibilistic logic in ILP for dealing with exceptions. *Artificial Intelligence*, 171(16-17):939–950, 2007.

[91] Mathieu Serrurier and Henri Prade. Improving inductive logic programming by using simulated annealing. *Information Scince*, 178(6):1423–1441, 2008.

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

[93] A. Srinivasan. The aleph manual, 1999.

[94] A. Srinivasan, R. King, and S. Muggleton. The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program, 1996. Under review for Intelligent Data Analysis in Medicine and Pharmacology. Kluwer Academic Press, 1996.

[95] A. Srinivasan, R. D. King, and D. W. Bristol. An assessment of ILP-assisted models for toxicology and the PTE-3 experiment. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634, pages 291–302. Springer-Verlag, 1999.

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

[97] 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.

[98] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Theories for mutagenecity: A study of first-order and feature based induction. Technical report, PRG-TR-8-95 Oxford University Computing Laboratory, 1995.

[99] R.W. Tennant, J. Spalding, S. Stasiewicz, and J. Ashby. Prediction of the outcome of rodent carcinogenicity bioassays currently being conducted on 44 chemicals by the national toxicology program. *Mutagenesis*, 5:3–14, 1990.

[100] S. D. Toprak. A new hybrid multi-relational data mining technique. Master's thesis, Middle East Technical University, Computer Engineering Department, Ankara, Turkey, May 2005.

[101] S. D. Toprak, P. Senkul, Y. Kavurucu, and I. H. Toroslu. A new ILP-based concept discovery method for business intelligence. In *ICDE Workshop on Data Mining and Business Intelligence*, April 2007.

[102] Ismail H. Toroslu and Meliha Yetisgen-Yildiz. Data mining in deductive databases using query flocks. *Expert Systems with Applications*, 28(3):395–407, 2005.

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

[104] Mahmut Uludag and Mehmet R. Tolun. A new relational learning system using novel rule selection strategies. *Knowledge-Based Systems*, 19(8):765–771, 2006.

[105] 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.

[106] P.R.J. van der Laag. *An Analysis of Refinement Operators in Inductive Logic Programming*. PhD thesis, Erasmus University, 1995.

[107] 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.

[108] I. Witten and et. al. Weka: Practical machine learning tools and techniques with java implementations. In Proc. ICONIP/ANZIIS/ANNES'99 International Workshop: Emerging Knowledge Engineering and Connectionist-Based Information Systems, pages 192-196, Dunedin, NZ, 1999.

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

[110] Milena Yankova and Svetla Boytcheva. Overview of inductive logic programming (ILP) systems. pages 345–359, 2002.

[111] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *ICDE*, pages 399–411, 2004.

APPENDIX A

IMPLEMENTATION DETAILS OF C^{2}D and CRIS

The implementation of both systems was done by using MS .NET C# programming language. To do this, MS Visual Studio .NET 2008 edition was used. The data sets were stored in a MS SQL SERVER 2005 database. Also, the other versions which run on MS Access, MySQL and Oracle were implemented.

In C^{2}D, the APRIORI lattice is composed of rules. Every rule has one head (target) and one or more body relations. Each relation has one or more arguments (parameters). Because of this, parameter, relation and rule are the three main classes in C^{2}D implementation.

The class definitions used in the C^{2}D algorithm are given in Table A.1, Table A.2 and TableA.3.

There is only one form (main form) in the implementation which is given in Figure 5.1. In the main form, first of all, the user selects the data set from the combo box, then the parameters (min_sup, min_conf, B, max_depth) are defined. The "Allow Recursion" checkbox is used to enable recursive rule search in C^{2}D and CRIS. However, "Consider Only Related Facts" checkbox is used only for C^{2}D to enable searching transitive rules. When the C^{2}D button is pressed, the C^{2}D algorithm starts. On the other hand, when the CRIS button is pressed, the CRIS algorithm starts. The important properties and methods in frmMain class are given in Table A.4.

A.1 Class Definitions in C^{2}D Algorithm

A.1.1 Parameter Class Definition

Table A.1: Parameter class

Prop/Meth	**Name:Type**	**Description**
Property	Name:string	Name of the parameter (either variable or constant name)
	ColName:string	Name of the column in the corresponding table
	TypeRel:string	Name of the type relation
	TypeCol:string	Name of the column in the type relation
	IsVar:boolean	A parameter is either a var or a const
	Sign:int	0: =, 1: ≤, 2: ≥
Method	Get/Set	Get and set methods for the properties
	IsEqual(parameter *p*):boolean	Returns true if the parameter is same as with *p*

A.1.2 Relation Class Definition

Table A.2: Relation class

Prop/Meth	**Name:Type**	**Description**
Property	Name:string	Name of the relation
	Parameters:array	An array containing parameters of the relation
Method	Get/Set	Get and set methods
	ParamCount:int	Returns number of params
	ContainParam(parameter *p*):int	Returns true if the relation contains *p*
	UpdateParam(int *idx*, string *newname*)	Updates the name of parameter at index (*idx*) as *newname*, also the parameter becomes a variable (if it is a constant)
	IsEqual(relation *r*):boolean	Returns true if the relation is the same as with *r*

A.1.3 Rule Class Definition

Table A.3: Rule class

Prop/Meth	Name:Type	Description
Property	Head:relation	Head of the rule
	Body:array	An array containing relations for the body
	NumConf:long	Numerator of the confidence
	DenConf:long	Denominator of the confidence
	NumSup:long	Numerator of the support
	DenSup:long	Denominator of the support
	NumF:long	Numerator of the f-score
	DenF:long	Denominator of the f-score
	NumConfP1:long	Numerator of the confidence of first parent rule
	DenConfP1:long	Denominator of the confidence of first parent rule
	NumConfP2:long	Numerator of the confidence of second parent rule
	DenConfP2:long	Denominator of the confidence of second parent rule
Method	Get/Set	Get and set methods
	BodyRelCount:int	Returns # of body relations
	AddBodyRelation(relation *r*)	Add *r* at the end of body
	RemoveBodyRelation(int *idx*)	Remove body reln at index index *idx*
	ContainParam(parameter *p*):boolean	Returns true if *p* exists in rule
	IsStrong():boolean	Returns true if confidence is greater than confidence of both parents
	CalculateConf()	Calculate NumConf and DenConf
	CalculateNumSup()	Calculate NumSup, DenSup is equal to number of uncovered instances which is calculated only once
	CalculateFScore(int *B*)	Calculate f-score value
	CoverExamples()	Cover the target instances which are deducible by the rule
	IsEqual(rule *cl*):boolean	Returns true if the rule is same as with *cl*

A.1.4 Main Form Class Definition

Table A.4: Main Form class

Prop/Meth	Name:Type	Description
Property	conn:connection	Database connection object
	targetDefn:relation	target relation
	factDefn:array	An array having background relations
	testDefn:relation	test relation (if exists)
	numUnEx:Long	Number of uncovered target instances
	min_sup:double	Minimum support bias
	min_conf:double	Minimum confidence bias
	B:int	B parameter of f-score value
	max_depth:int	Maximum number of body relations
	allowRec:boolean	Allow recursion parameter
	onlyRelatedFacts:boolean	Use only related facts parameter (used in basic version of C^2D)
	factRules:array	An array having fact rules for each coverage
	candidateRules:array	An array having candidate rules after pruning according to min_sup criteria for each level in APRIORI lattice
	allCandidateRules:array	An array having all strong candidate rules after pruning according to min_conf criteria before each coverage step
	solutionRules:array	An array having solution rules (having max f-score value) before each coverage step
Method	HasCommonParam (relation *r1*, relation *r2*:boolean)	Returns true if *r1* and *r2* are related (have parameters with same type)
	RemoveERPrimForeRules	Removes rules which do not obey primary-foreign key relations
	AddIndirectlyRelatedFacts()	Add indirectly related background facts to factRules
	CretaGeneralRules()	Finds two literal candidate rules
	SpecializeOneStep()	Finds candidate rules for next level
	FindSolutionRules()	Finds solution rules in the APRIORI lattice and cover target instances deducible by solutionRules

A.2 Pseudocode of Functions in C^{2}D and CRIS Algorithms

The pseudocode of the C^{2}D and CRIS algorithms with their important functions are given in Tables A.5, A.6, A.7, A.9 and A.10

A.2.1 Pseudocode of Main Function in C^{2}D Algorithm

Table A.5: The pseudocode of main C^{2}D function

```
initialize: Initializes variables and get input variables from gui
initializeUncovered: Calculate factRules
while (numUnEx > 0 AND selected target instance index is smaller than numUnEx)
    generalize: For each fact rule,
                find most general candidate rules (one head - one body literal)
    If there is no candidate rule, select next target instance
    Otherwise;
        while (currentDepth < maxDepth)
            specializeOneStep: Create candidate rules
                               for the next level of APRIORI lattice
            current-depth = current-depth + 1
        If there is no candidate rule, select next target instance
        Otherwise;
            findSolutionRules: Find the best rules
            For each best rule;
                cover the target instances captured by the rule
                If there is a test relation,
                    then cover the test relation instances captured by the rule
    initializeUncovered: Calculate factRules
Find the coverage (accuracy) values of the hypothesis for the test relation if exists
```

A.2.2 Pseudocode of InitializeUncovered Function

Table A.6: The pseudocode of initializeUncovered function

```
getUncoveredExamples: Select uncovered target instances from DB by SQL query and
                                    assign numUnEx as count
if (numUnEx == 0) then all target instances are covered, stop
if (selected target instance index > numUnEx) then
    There are no more possible rules. Stop
Select a target instance and define head relation
if (recursion is allowed) then
    Find related facts from target relation
    For each related fact
        Define body relation
        Add body relation to head to define a rule and put it into factRules
For each background relation
    Find related facts from the selected background relation
    For each related fact
        Define body relation
        Add body relation to head to define a rule and put it into factRules
If Add Indirectly Related Facts option is checked
    For each fact rule
        Find related facts of the body relation
        For each related fact
            Define indirectly related body relation
            Add indirectly related body relation to head
                    to define a rule and put it into factRules
```

A.2.3 Pseudocode of Generalize Function in C^{2}D

Table A.7: The pseudocode of generalize function in C^{2}D

```
Calculate possible different head relations for the selected target instance
            (variable/constant possibilities for each argument)
For each possible head relation
    For each rule in FactRules
        Get the body fact relation
        Calculate possible different body relations for the selected body fact relation
            (variable/constant possibilities for each argument)
        Add each possible body relation to head and define a candidate rule
        Add the new candidate rule to candidateRules
Prune the rules in candidateRules which do not obey the
        primary-foreign key relationship (if exists) property
Calculate support values for each candidate rule
Prune the candidate rules having support less than min_sup
Calculate confidence values for each frequent candidate rule
Add strong candidate rules to the allCandidateRules
            (having confidence greater than min_conf)
```

A.2.4 Pseudocode of Generalize Function in CRIS

Table A.8: The pseudocode of generalize function in CRIS

```
Calculate possible different constants for each argument of the head relation
            (and generate possible head relations)
For each possible head relation
    For each relation in Background Knowledge
        Get the background fact relation
        Calculate possible different constants for each argument of the
            selected fact relation (and generate possible body relations)
        Add each possible body relation to head and define a candidate rule
        Add the new candidate rule to candidateRules
Prune the rules in candidateRules which do not obey the
        primary-foreign key relationship (if exists) property
Calculate support values for each candidate rule
Prune the candidate rules having support less than min_sup
Calculate confidence values for each frequent candidate rule
Add strong candidate rules to the allCandidateRules
            (having confidence greater than min_conf)
```

A.2.5 Pseudocode of SpecializeOneStep Function

Table A.9: The pseudocode of specializeOneStep function

```
For each rule in the candidateRules list except the last one
    Store the current candidate rule in variable rule1
    For each rule in the candidateRules list
                from the next of rule1 through the last one
        Store the iterated candidate rule in variable rule2
        if (head(rule1) = head(rule2) then
            unionOneStep(rule1,rule2): Find candidate child rules
Calculate support values for each candidate rule
Prune the candidate rules having support less than min_sup
Calculate confidence values for each frequent candidate rule
Prune the candidate rules having confidence value
                not greater than the parents' confidence values
Add strong candidate rules to the allCandidateRules
                (having confidence greater than min_conf)
```

A.2.6 Pseudocode of UnionOneStep Function

Table A.10: The pseudocode of unionOneStep function

```
INPUT: rule1, rule2
unionOneStep(rule1,rule2,bodyIdx1,bodyIdx2):
                    returns the different body relation index of rule1 into bodyIdx1 and
                    the different body relation index of rule2 into bodyIdx2
                    if rule1 and rule2 have only one different body relation
                    otherwise bodyIdx1 will be -1
if (bodyIdx1 == -1) then stop
Store rule1.getBodyRelation(bodyIdx1) into bRel1 and
                    rule2.getBodyRelation(bodyIdx2) into bRel2
Find number of different variable arguments in rule1
For each argument in bRel2
    if the current argument is not a variable do nothing
    update the variable name of current argument
                        according to number of variables in rule1
    update the following variable argument names
                        if they do not exist in other relatons
Add bRel2 into possible new body relation list
For each argument in new bRel2
    if the current argument is not a variable do nothing
    if the current argument exists in the head or other body relations do nothing
    For each argument in bRel1
        if the current argument is not a variable do nothing
        if the current argument exist in the head do nothing
        if the current argument has same type with the selected argument of bRel2
            create a new body relation (copy of bRel2)
            change the argument with the name of current argument of bRel1
            update the following arguments of new relation if needed
            add the new relation into possible new body relation list
For each relation in possible new body relation list
    define a new rule as possible relations added to end of rule1 body
    set parents' confidence values of the new rule
    according to rule1 and rule2 confidence values
    add the new rule into candidate rule list
```

APPENDIX B

THE SQL STATEMENTS FOR EACH ARGUMENT IN PTE_ATM RELATION

The feasible constants and the SQL statements used for each argument of *pte_atm* are as shown in Table B.1.

Table B.1: The feasible constants for pte_atm relation

Argument	**Constants**	**SQL Query**
drug	empty set (only variable)	SELECT drug FROM pte_atm GROUP BY drug HAVING COUNT(*) ≥ 9189 * 0.05
atom	empty set (only variable)	SELECT atom FROM pte_atm GROUP BY atom HAVING COUNT(*) ≥ 9189 * 0.05
element	c, h, o (also variable)	SELECT element FROM pte_atm GROUP BY element HAVING COUNT(*) ≥ 9189 * 0.05
integer	3, 10, 12 (also variable)	SELECT integer FROM pte_atm GROUP BY integer HAVING COUNT(*) ≥ 9189 * 0.05
charge	19 range constants (also variable)	SELECT charge FROM pte_atm ORDER BY charge

VITA

PERSONAL INFORMATION

Surname, Name:	Kavurucu, Yusuf
Nationality:	Turkish (TC)
Date and Place of Birth:	24 November 1978, Aydın
Marital Status:	Married
Phone:	+90 312 402 2273
e-Mail:	ykavurucu@yahoo.com

EDUCATION

Degree	Institution	Year of Graduation
MS	METU-Computer Engineering	2002
BS	METU-Computer Engineering	2000
High School	Aydın Adnan Menderes Anatolian HS	1996

WORK EXPERIENCE

Year	Place	Enrollment
2007-present	CIS Department, Turkish General Staff HQ	System Engineer
2000-2007	CIS Department, Turkish Naval Forces HQ	Software Developer

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Kavurucu Y., Senkul P., Toroslu I.H., “Multi-Relational Concept Discovery with Aggregation”, 24^{th} International Symposium on Computer and Information Sciences (ISCIS), Northern Cyprus, September 2009
2. Kavurucu Y., Senkul P., Toroslu I.H., “ILP-based Concept Discovery in Multi-Relational Data Mining”, Journal of Expert Systems with Applications, 36(9):11418-11428, November 2009

3. Kavurucu Y., Senkul P., Toroslu I.H., “Analyzing Transitive Rules on a Hybrid Concept Discovery System”, 4^{th} International Conference Hybrid Artificial Intelligent Systems (HAIS) Salamanca, Spain, June 2009
4. Kavurucu Y., Senkul P., Toroslu I.H., “Confidence-based Concept Discovery in Relational Databases”, 2009 World Congress on Computer Science and Information Engineering (CSIE 2009) Los Angeles, USA, April 2009
5. Kavurucu Y., Senkul P., Toroslu I.H., “Aggregation in Confidence-based Concept Discovery for Multi-Relational Data Mining”, IADIS European Conference on Data Mining (ECDM), pages 43-50, Amsterdam, Netherlands, July 2008
6. Kavurucu Y., Senkul P., Toroslu I.H., “Confidence-based Concept Discovery in Multi-Relational Data Mining”, International Conference on Data Mining and Applications (ICDMA), pages 446-451, Hong Kong, March 2008
7. Toprak S.D., Senkul P., Kavurucu Y., Toroslu I.H., “A New ILP-based Concept Discovery Method for Business Intelligence”, International Conference on Data Engineering (ICDE) Data Mining and Business Intelligence Workshop, İstanbul, Turkey, April 2007