IMPROVING SCALABILITY AND EFFICIENCY OF ILP-BASED AND GRAPH-BASED CONCEPT DISCOVERY SYSTEMS

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

ALEV MUTLU

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

JANUARY 2013

Approval of the thesis:

IMPROVING SCALABILITY AND EFFICIENCY OF ILP-BASED AND GRAPH-BASED CONCEPT DISCOVERY SYSTEMS

submitted by **ALEV MUTLU** 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. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Pınar Karagöz _____
Supervisor, **Computer Engineering Department, METU**

Examining Committee Members:

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

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

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

Assist. Prof. Dr. Osman Abul
Computer Engineering Department, TOBB ETÜ _____

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: ALEV MUTLU

Signature :

ABSTRACT

IMPROVING SCALABILITY AND EFFICIENCY OF ILP-BASED AND GRAPH-BASED CONCEPT DISCOVERY SYSTEMS

Mutlu, Alev

Ph.D., Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Pınar Karagöz

January 2013, 96 pages

Concept discovery is the problem of finding definitions of target relation in terms or other relation given as a background knowledge. Inductive Logic Programming (ILP)-based and graph-based approaches are two competitors in concept discovery problem. Although ILP-based systems have long dominated the area, graph-based systems have recently gained popularity as they overcome certain shortcomings of ILP-based systems.

While having applications in numerous domains, ILP-based concept discovery systems still sustain scalability and efficiency issues. These issues generally arose due to the large search spaces such systems build. In this work we propose memoization-based and parallelization-based methods that modify the search space construction step and the evaluation step of ILP-based concept discovery systems to overcome these problem.

In this work we propose three memoization-based methods, called *Tabular CRIS*, *Tabular CRIS-wEF*, and *Selective Tabular CRIS*. In these methods, basically, evaluation queries are stored in look-up tables for later uses. While preserving some core functions in common, each proposed method improves efficiency and scalability of its predecessor by introducing constraints on what kind of evaluation queries to store in look-up tables and for how long.

The proposed parallelization method, called *pCRIS*, parallelizes the search space construction and evaluation steps of ILP-based concept discovery systems in a data-parallel manner. The proposed method introduces policies to minimize the redundant work and waiting time among the workers at synchronization points.

Graph-based approaches were first introduced to the concept discovery domain to handle the so called

local plateau problem. Graph-based approaches have recently gained more popularity in concept discovery system as they provide convenient environment to represent relational data and are able to overcome certain shortcomings of ILP-based concept discovery systems. Graph-based approaches can be classified as structure-based approaches and path-finding approaches. The first class of approaches need to employ expensive algorithms such as graph isomorphism to find frequently appearing substructures. The methods that fall into the second class need to employ sophisticated indexing mechanisms to find out the frequently appearing paths that connect some nodes in interest. In this work, we also propose a hybrid method for graph-based concept discovery which does not require costly substructure matching algorithms and path indexing mechanism. The proposed method builds the graph in such a way that similar facts are grouped together and paths that eventually turn to be concept descriptors are build while the graph is constructed.

Keywords: Inductive Logic Programming, Graph, Concept Discovery, Scalability, Efficiency

ÖZ

TÜMEVARAN MANTIK PROGRAMLAMA TABANLI VE ÇİZGE TABANLI KAVRAM KEŞİF SİSTEMLERİNİN ÖLÇEKLENDİRİLEBİLİRLİK VE VERİMİNİN ARTIRILMASI

Mutlu, Alev

Doktora, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Doç. Dr. Pınar Karagöz

Ocak 2013 , 96 sayfa

Kavram keşfi, bir ilişki tanımının arkaplan ilişlileri olarak adlındırılan diğeri bazı ilişkiler cinsinden bulunması problemidir. Kavram keşif probleminde Tümevaran Mantık Programlama (TMP) ve çizge tabanlı yöntemler yoğun olarak kullanılmaktadır.TMP tabanlı yaklaşımlar bu problemde uzun süredir hakim bir şekilde kullanılıyor olsa da, bu tür sistemlerin bazı problemlerini çözdüğü için çizge tabanlı yaklaşılar da son zamanlarda bu alanda popülarite kazanmıştır.

Birçok alanda uygulaması bulunmakla birlikte, TMP tabanlı sistemlerde verimililik ve ölçeklendirilebilirlik sorunları mevcuttur. Bu sorunlar genellikle TMP tabanlı sistemlerin oluşturduğu büyük arama alanlarından kaynaklanmaktadır. Bu çalışmada, tablolama ve paralleleştirme yöntemleri kullanılarak TMP tabanlı kavram keşif sistemlerinin arama alanı oluşturma ve değerlendirme aşamalarının iyileştirilmesi için yöntemler sunulmaktadır.

Bu çalışmada tablolama yöntemlerini kullanan üç yöntem sunulmuştur, *Tabular CRIS*, *Tabular CRIS-wEF* ve *Selective Tabular CRIS*. Bu metotlarda, temel olarak, arama alanı değerlendirme sorguları daha sonra kullanılmak üzere tablolanmaktadır. Her üç metot bazı ortak temel fonksiyonları barındırmakla birlikte, ardıl gelen öncekinin ölçeklendirilebilirliğini iyileştirmek için bazı yeni yaklaşımlar sunmaktadır.

Önerilen paralel yöntem, *pCRIS*, TMP tabanlı sistemlerin arama alanı oluşturma ve değerlendirme aşamalarını veri-paralel yöntemler kullanılarak paralelleştirmektedir. Önerilen yöntem fazladan yapılan işi ve senkronizayon anlarında işlemcilerin bekleme süresini asgari düzeyde tutacak şekilde tasarlanmıştır.

Çizge tabanlı sistemler öncelikle TMP tabanlı sistemlerin problemlerinden biri olan yerel plato prob-

lemini gidermek için sunulmuştur. Veriyi etkin bir şekilde ifade edebildiği ve TMP tabanlı sistemlerin bazı problemerini ortadan kaldırdıgı için yakın zamanlarda, kavram keşfi probleminde, çizge tabanlı sistemler artan bir popülarite kazanmaktadır. Genel olarak bu tür sistemler ortak bileşen bulma ve yol bulma sistemleri olarak gruplandırılabilir. İlk kümeye düşen yöntemler ortak bileşenleri bulmak pahalı algoritmalar kullanmaktadır. İkinci kümeye düşen yaklaşımlar ise çizge içideki yolları tumak için komplike indeksleme yöntemlerine ihityaç duymaktadır. Bu çalışmada çizge tabanlı kavram keşif sistemleri için pahalı bileşen bulma algoritmalarına ve komplike indeksleme mekanizmalarına ihtiyaç duymayan melez bir yöntem sunulmaktadır. Önerilen yöntemde bezer yapılar gruplanmakta ve kavram tanımlarını oluşturan yollar çizge inşaa edilirken oluşturulmaktadır.

Anahtar Kelimeler: Tümevaran Mantık Programlama, Çizge, Kavram Keşfi, Ölçeklendirilebilirlik, Verimlilik

To my parents and grandparents

Emine, Niyazi, Ayşe, Ömer

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Professor Pınar Karagöz for giving me the opportunity to work with her. This dissertation would not be possible without her constructive criticism, positive attitude and willingness to help. She has always been a great resource for me not only in research issues, but also in many matters of life. I feel really lucky to be her advisee and once more would like to express my greatest thanks.

I would like express my thanks to Professor İsmail Hakkı Toroslu and Professor Osman Abul for serving as members of my thesis progress committee. Their comments were of great value and their positive attitude have always motivated me.

I would like to thank Professor Ahmet Coşar and Professor Tolga Can for accepting to be members of my dissertation committee. Their valuable feedback was of great value to finalize this work.

I really owe much to Dr. Yusuf Kavurucu. He supplied me with the material for the start up of this work. I would like to express my greatest thanks to him for his continuous support and contributions to this work.

I would like to express my thanks to Professor Meral Özsoyoğlu and Professor Tekin Özsoyoğlu for their guidance and support during my research visit to Case Western Reserve University, Cleveland, OH.

I would like to acknowledge my friends. I would like especially thank to Levent, Gülşah, Ruken, Özgür, Semra, Burçak, Derya, Özge, Sinan, Sarp, Umut, Hasibe and Xinjian. They made my life fun. I would to express my special thanks to Çelebi and Ahmet for their great friendship and valuable help with the department's high performance computing system, Nar.

Finally, my deepest thanks go to my family. I would like to thank my parents Emine and Niyazi, and grandparents Ayşe and Ömer, for their love, support and patience. They have always made their best to provide me with a convenient environment and were the greatest supports of mine in all matters of life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Learning from data has always been a challenging research topic in computer science. With the evolution of data collection and storage tools, the form of learning has also evolved, i.e. from propositional learners that work data stored in flat tables [126] to relational learners that work on data that is stored in relational databases [127].

Inductive Logic Programming (ILP) [107] provides a conventional environment to induce user interpretable patterns that define data stored in relational databases. One of the mostly addressed tasks in ILP is concept discovery [54] where the problem is to find theories that define a specific target relation in terms of other relations provided as background data. Although ILP-based concept discovery systems have been applied in various domains with promising experimental results [55, 160, 8], they have efficiency and scalability issues. Methods such as query transformations [41], limiting the search space [156] have been proposed to handle these issues. Another frequently faced problem with ILP-based concept discovery systems is the local plateau issue [133], where refinement operators of ILP are insufficient to improve concept descriptor's quality. To cope with this problem graph-based approaches have been introduced. Recently graph-based approaches [66, 69, 164] have become a competitor to ILP-based approaches in concept discovery problem.

In this dissertation we propose methods to improve scalability and efficiency issues of ILP-based concept discovery systems. The proposed methods are based on memoization and parallelization. Those methods are implemented as extensions to the ILP-based predictive concept discovery system called CRIS [81, 83].

In addition to this, we also present a graph based concept discovery algorithm and investigate the effect of the graph based model and solution on the execution time. For graph-based approach, a subset of concept discovery problem that includes only predicates with two attributes are considered. The graph based concept discovery algorithm proposes a new representation framework for data, which is more compact and human interpretable. The graph based algorithm constructs the concept descriptors while building the graph from data and does not require any further graph operations to construct the concept descriptors.

In this chapter we firstly prove a formal definition of ILP-based concept discovery. Later, in Section 1.1, we discuss our motivation. In Section 1.2, we briefly introduce the proposed methods and list our contributions. The last section presents the outline of this dissertation.

1.1 Problem Definition and Motivation

Inductive Logic Programming is a research area at the intersection of machine learning and logic programming. It provides powerful tools to infer general patterns that define facts which is stored in relational format. ILP systems can be classified as descriptive learners or predictive learners. Descriptive ILP systems look for any pattern that can be considered valid based on some user defined metrics. Predictive ILP systems, which are also called concept discovery systems, look for patters that define a certain relation in terms of other relations available in the data set. Those auxiliary data is called background knowledge.

ILP-based concept discovery systems input a set of target instances, a set of background knowledge and some user-defined criteria to evaluate the concept descriptors and output solution clauses in the form of Horn clauses [127]. ILP-based concept discovery systems can be as considered as supervised learners as the target instances are labeled as *positive* if they belong to the target concept, or *negative* if they do not belong to the target concept. An intermediate hypothesis is called *strong* if it does not cover all the positive examples, and *weak* if it covers some negative examples. Such hypotheses are refined to form the final hypothesis set whose members are expected to be *complete* and *consistent*. The induced set of hypothesis is complete if every positive target instances are modeled by at least one hypothesis, and consistent none of the negative target instances are explained by any hypothesis.

Although such systems have been applied in several domains and promising results have been reported, they still sustain efficiency and scalability issues. These problems are mainly due to the evaluation of large search space such systems build. Costa et. al. [41] reported that almost 90% of the total execution time of ALPEH [150] is spent on evaluating the search space when run on the Muta data set, and around 80% of the total execution time when run on the PTE data set. Similar results have also been reported for WARMR by Blockeel et. al. [21]. Several studies have been proposed to improve this step such as modifying evaluation mechanism [41, 40], putting constraints on structure of allowable concept descriptors [156]. Although computations in search space formation are inexpensive, this step may require long running time if there exist to many concept to be considered.

In this study we work on the efficiency and scalability issues of ILP-based concept discovery systems from two different points. We propose methods that incorporate memoization techniques to improve the running time of the search space formation step and the search space formation step of ILP-based concept discovery systems. Applications that employ memoization are vulnerable to memory overflow problems. In this study we also propose a method that utilizes the memory requirement of ILP-based concept discovery systems that employ memoization.

The second approach we follow to improve running time of ILP-based concept discovery is parallelization. We propose a parametric system that parallelizes the search space formation and the search space evaluation steps of the ILP-based concept discovery systems. In the proposed parallelization method user can determine which parts of the ILP-based concept discovery system to run in parallel, minimum work load required to run the components in parallel, and size of the chunks.

Graphs are powerful data structures to represent relational data and provide tools to induce patterns that are valid within the graph representation of the data. As concept discovery systems work on large data sets, representing each fact as a distinct object, i. e. by a single vertex or a group of vertices, in the graph and employing purely graph oriented approaches to discover patterns may be computationally expensive, and are hard to be interpret by users. In this study we propose a method for compact representation of data on a graph structure. The proposed method builds the potential concept descriptors while building the graph which enables the proposed approach to avoid costly

graph substructure discovery process.

1.2 Contributions

In this study we address the scalability and efficiency issues of ILP-based systems and introduce memoization and parallelization based methods to improve them. In addition to this, we also elaborate the representation and efficiency issues of graph-based concept discovery systems and propose initial results of a hybrid graph-based concept discovery system. In this section we briefly introduce our proposed methods and our contributions in turn in order to this section be self-contained.

In this dissertation we propose three methods that incorporate memoization, namely *Tabular CRIS*, *Tabular CRIS-wEF*, and *Tabular CRIS-S*.

Tabular CRIS is proposed to deal with the repeating queries that are generated within the same iteration. It stores the evaluation queries and the number of tuples returned by these queries in look-up tables. If a query is regenerated within the same epoch, its result is directly retrieved from the look-up tables.

Tabular CRIS-wEF is proposed to deal with handling the repeated queries that are generated both within the same and different iterations. In case of noisy and incomplete data, ILP-based concept discovery systems perform multiple runs on the data and at the end of each iteration target instances explained by the currently induced rules are removed from the data set. Tabular CRIS-wEF stores the tuples returned by queries. Tabular CRIS-wEF modifies the set covering method in such a way that it does not only removes the explained target instances from the data set but also from the look-up tables. By this way the look-up tables store the updated results of evaluation queries, and result of a query that is generated in a previous iteration can be retrieved from the look-up table.

Although Tabular CRIS-wEF improves the look-up table hit count of Tabular CRIS in a great extent, it introduces memory problems. Although one can not know if a certain evaluation query will be regenerated in a later step, one can know that a certain evaluation queries will not be. Tabular CRIS-S proposes policies on what type of evaluation queries to store in look-up tables and for how long. Tabular CRIS-S also modifies the search space formation step of Tabular CRIS for further memory utilization.

Although memoization based techniques have widely been studied in ILP-based concept discovery community, the proposed methods have certain advantages over the reported studies. These can be listed as follows:

- The proposed approaches focus on the similarity among evaluation queries, while most of the studies such as [21] focus on the similarity among concept descriptors. Even two concept descriptors differ in structure, such concept descriptors may map into the same evaluation query. Such cases can not be handled by looking for similarities among concept descriptors.
- Many of the approaches that employ memoization such as [137] look for similarities of the concept descriptors that are of the same length. Concept descriptors of different lengths may share the same evaluation query if the longer one differs from the shorter one with literals that contain unbounded variables. The proposed approaches can handle such situations.
- Due to the large search space and size of the data sets, ILP-based concept discovery systems that embody memoization usually face memory overflow problems. Several studies implement special purpose data structures such as [64] to cope with such issues. Tabular CRIS-S implements

look-up table cleaning mechanisms to remove the evaluation queries and their associated results that are known not to be regenerated.

- Many of the reported attempts modify the underlaying data storage engine, which is Prolog in most of the cases, to handle the repeating queries. Our approach is coded within the induction system, which makes it transparent to the user and the underlaying data management and storage engine.

ILP-based concept discovery systems are well amenable for parallel implementation as their most time consuming step, the search space evaluation step, consists of executing several evaluation queries that perform read-only operations on the data set. Write operations are only performed at the end of each iteration to remove the explained target instances from the data set. The proposed parallelization method, called *pCRIS*, employs master-worker architecture and implements data-parallelization. pCRIS dispatches evaluation queries among multiple computation units and gathers the results to prune the search space.

Several studies have been proposed to parallelize the search space evaluation step of ILP-based concept discovery systems. Other than the parallelization techniques, such systems differ in what they split among the computational units. A class of such systems split the background data among the computational units and run the same query on smaller data sets and sum up the partial results to calculate the global solution [98]. Another class of parallel ILP-based concept discovery systems split the target instances among the computational units, each computational unit then runs an ILP-concept discovery system to induce concept descriptors that define its share of target instances [45]. In the former class of approaches, evaluation queries may be run on irrelevant data portions[1], while in the later approach different computational units may run a number of identical queries.

In pCRIS a certain node, called *master*, generates the evaluation queries and dispatches them among multiple workers each of which has access to the entire data set. By this way pCRIS avoids the risk of running an evaluation query on irrelevant data set. The master also maintains a list of the dispatched evaluation queries in a look-up table, and before sending a new evaluation query to a worker it first scans it in the list to find out if the query is already sent to some worker. If the scan results with a hit, the query is not sent to the worker but its result is retrieved from the look-up. By this way pCRIS avoid execution of repeating queries.

pCRIS also proposes parallelization method to improve running time of search space formation step of ILP-based concept discovery systems. In the search space formation step, each hypotheses in the current search space is compared to the every other concept descriptor and unifiable ones are merged to form the search space of the next level. pCRIS splits the current search space among multiple computational units in such a way that each worker extends mutually exclusive subsets of the search space and sends the resulting partial search spaces to the master to form the global search space.

pCRIS has significant advantages over reported studies. These can be listed as follows:

- pCRIS is a parametric system. User can set which steps to run in parallel.
- User can provide pCRIS with minimum work load for parallel execution. This allows pCRIS to execute certain functions in parallel when work load is high, otherwise in sequential manner. By this way pCRIS prevents the communication cost overwhelm the computational cost.

[1] We call a portion of a data set irrelevant with respect to the evaluation query if the portion does not contain some of relations with in the query.

- While dispatching the evaluation queries among works, pCRIS calculates the size of chunks dynamically in a monotonically decreasing order. By this way, pCRIS minimizes the waiting time for the last computational node to finish it calculations.
- pCRIS maintains a central look-up table to avoid execution of the repeating queries.

In this work, we also introduce preliminary results of a new graph based concept discovery system. The proposed method can be considered as a hybrid graph-based concept discovery system as it modifies the representation of data in graph structure and induces concept descriptors using path-finding techniques. In graph-based concept discovery systems, generally, a vertex represents a fact or an attribute of a fact. Edges connect such related vertices. To induce concept descriptors, such systems either look for frequently appearing substructures, which is the case with substructure-based approaches, or paths that connect certain type of vertices, which is the case with path finding-based approaches. Discovering frequently appearing substructures requires employing expensive graph operations such as graph isomorphism. Looking for paths that connect certain type of vertices requires complex indexing mechanism. In addition, when data is large, representing each fact or an argument of a fact as a distinct vertex results in large graphs, which may be difficult for human interpretation.

We propose to store the facts that are *similar* to each other as a single vertex in the graph. In this work, two facts are considered *similar* if they are related to the same fact with the same relation. Edges are labeled after the relation name endpoints are involved with each other. Each vertex in the graph also stores the its reachability information, i. e. the path to follow, to any other vertex within the graph that it is related to.

We call the proposed approach hybrid as:

1. In context of substructure-based approach, vertices mimic frequently appearing substructures as they hold information for similar facts.
2. In context of path-finding approach, to induce the concept descriptors from such constructed graph, one only needs the reachability information of the vertices he is interested in.

The quality of these concept descriptors are calculated by translating them into SQL queries and executing them on the database management engine.

The advantages of the proposed graph-based concept discovery can be listed as follows:

- It provides a compact representation of the data which makes it easier for human interpretation.
- It does not require expensive graph substructure matching algorithms to find similar substructures as graph is constructed in such a way that similar objects are represented as a single vertex.
- It does not require searching for paths that connect some vertices, but reduces the problem to searching the graph for the vertex is in interest and retrieving its reachability information to the other vertices.

1.3 Organization of the Thesis

This dissertation is composed of seven chapter. Chapter 1 introduces the problem definition and the motivation behind this dissertation. Chapter 2 presents the related work on parallelization in ILP-based

concept discovery. In this section we also present the related work on memoization-based approaches to speed-up ILP-based concept discovery systems. Related work on graph based concept discovery is also presented in Chapter 2. Chapter 3 introduces Tabular CRIS, Tabular CRIS-wEF, and Tabular CRIS-S, the memoization-based methods. Chapter 4 introduces parallel version of CRIS, pCRIS. Chapter 5 presents the proposed graph based concept discovery system. Chapter 6 presents the experimental results. Experimental results are discussed in comparison to the original implementation of CRIS and several other state of the art studies. Chapter 7 concludes the dissertation with future directions.

CHAPTER 2

BACKGROUND

In this chapter we firstly introduce ILP-based concept discovery in general and then explain the predictive ILP-based concept discovery system Concept Rule Induction System (CRIS) in detail as it is the basis for the proposed methods studied in this thesis. In the subsequent sections we summarize related work aiming to improve ILP-based concept discovery systems in terms of memoization and parallelization. Lastly we present related work on graph-based concept discovery.

2.1 Overview on Inductive Logic Programming

With the increasing amount of data collected in relational databases, the need for mining algorithms that can work on structured data has emerged. One proposed approach [88] is to convert database tables into a single table and run propositional learners such as C4.5 [129]. However, integrating data from multiple tables into a single table may result in the loss of information and is not desirable. Another approach [93] is to convert propositional learners into versions that directly work with relational databases. Many classical propositional learners have their relational versions, e.g., relational decision tree TILDE [23] is an upgrade of C4.5.

Many of the learning algorithms that use relational databases, called Multi Relational Data Mining (MRDM) algorithms, have their roots in Inductive Logic Programming (ILP) [54]. ILP [107] is a research area at the intersection of Machine Learning and Logic Programming and provides tools that infer patterns that are valid for a given set of facts. Such systems usually employ first order logic as the representation language for the input data, and represent the induced patterns in the form of Horn clauses. In Table 2.1 we provide the database terminology and its counterpart in Inductive Logic Programming terminology.

Table2.1: Data base and logic programming terminology

DB Terminology	**LP Terminology**
relation name p	predicate symbol p
attribute of relation p	argument of predicate p
tuple $a_1, ..., a_n$	ground fact $p(a_1, ..., a_n)$
relation p a set of tuples	predicate p defined extensionally by a set of ground facts
relation p defined as a view	predicate p defined intensionally by a set of clauses

ILP systems are divided into two categories based on what they learn: *predictive systems* and *descrip-*

tive systems. In predictive ILP systems there is a specific target concept to be learned; however there is no specific goal in descriptive ILP learning [67].

One of the most commonly addressed task in multi relational learners is concept discovery [54], which falls into the category of predictive learning tasks in context of ILP. ILP-based concept discovery systems input a set of target instances, a set of background knowledge, and criteria to evaluate the induced concept descriptors. Such systems, similar to the framework of MRDM, generally employ first order logic to represent the data and output the concept descriptors in the form of Horn clauses where the negated literals are from the background data and the positive literal is the target relation. If recursion is supported, the target relation may participate as a negative literal in the induced Horn clause.

Two criteria to evaluate the induced patterns in ILP are *completeness* (Formula 2.1) and *consistency* (Formula 2.2). A pattern is called complete if it explains all the positive target instances, and consistent if it explains none of the negative target instances.

$$complete(B, H, E) = \{e \in E^{+} | B \cup H \models e\} \tag{2.1}$$

$$consistent(B, H, E) = \{e \in E^{-} | B \cup H \models \emptyset\} \tag{2.2}$$

These two metrics are generally quite hard to achieve as the data may contain noise or may be incomplete [108]. For this reason, in ILP-based concept discovery systems, definitions of completeness and consistency are relaxed to cover as many positive target instances as possible, and as few negative target instances as possible, respectively.

ILP-based concept discovery systems generally start with an initial hypothesis and refine it by one literal at a time until they meet the quality metrics. At each iteration the intermediate concept descriptors are tested against the background data, and based on their quality they are either pruned, further refined in the next iteration or added into the solution set. If at the end of the iteration some concept descriptors are added into the solution set, target instances they explain are removed from the target instance set or are marked as covered. In cases where the input data set is incomplete or noisy, an ILP-based concept discovery system needs to loop through these steps, namely *initial hypothesis generation*, *refinement*, *evaluation*, and *covering*, multiple times until all target instances are covered or no more concept descriptors can be induced that explain the remaining target instances. Algorithm 1 outlines a generic ILP-based concept discovery system.

Based on the search direction, ILP-based concept discovery systems are classified as *top-down* systems and *bottom-up* systems. Top-down systems start with a set of overly general initial theories that cover both positive and negative target instances and refine them until they are complete and consistent. In the context of top-down ILP-based concept discovery systems refinement operators are called *specialization* operators. Concept descriptors are specialized by adding a literal to the body of the concept descriptor, or applying substitution to the concept descriptors. Such refinement operators include a refinement graph which is used in MIS [145], and Apriori which is used in WARMR [44].

In bottom-up concept discovery systems, the initial hypothesis is very specific, called the *bottom clause*, and models only one positive target instance. In such systems the concept descriptors are refined by one literal at a time until they meet the quality measures. Generalization of a bottom clause is achieved by removing a literal from the body of the concept descriptor, or applying inverse substitution to the concept descriptor. Such refinement operators, called *generalization operators*, include

Algorithm 1 Generic ILP-based concept discovery

Require: E: target instances, B: Background knowledge
Ensure: H: Complete and consistent concept descriptors

```
1:  H'' = ∅
2:  while E ≠ ∅ or H'' ≠ ∅ do
3:     H'' = ∅
4:     Start with an initial hypothesis set H'
5:     for all h ∈ H' do
6:        refine(h)
7:        evaluate(h, B)
8:        if good(h) then
9:           cover(E, h)
10:          H'' = H'' ∪ h
11:       end if
12:    end for
13:    H = H ∪ H''
14: end while
15: RETURN H
```

relative least general generalization (rlgg) which is used in GOLEM [110], and inverse resolution which is used in CIGOL [109].

There are also hybrid concept discovery systems, such as CRIS [83], that incorporate properties of both top-down and bottom-up concept discovery systems.

2.1.1 Overview of ILP-based Concept Discovery Systems

ILP-based concept discovery systems have applications in a wide range of domains. In this section, we introduce some of the general purpose, state of the art ILP-based concept discovery systems and provide real-life domains they have been applied on.

WARMR [44, 46] is a top-down descriptive ILP system. It employs APRIORI rule [12] as the search heuristic to traverse the search lattice. It requires mode declarations [156] to constrain the search space. WARMR employs support and confidence values to prune the concept descriptors. As being a descriptive ILP system, it finds frequent patterns instead of concept descriptors. WARMR may mimic a concept discovery system by outputting frequent patterns that contain a specific target relation instead of all frequent patterns hidden in the data. WARMR has been applied on problems such as the carcinogenesis prediction [84] and gene function prediction [36].

GOLEM is a bottom-up ILP-based concept discovery system. It utilizes relative least general generalization operator [110] to constrain the search space. It requires mode declarations and negative target instances. GOLEM can not induce recursive concept descriptors. GOLEM has been applied on problems such as the finite element mesh design [49] and protein secondary structure prediction [111].

CLAUDIEN [130] is a top-down ILP-based system that searches the search space in an iterative deepening manner where clauses are expanded by clausal refinement operator. A clause is pruned in CLAUDIEN if it is implied by the current clausal theory and if it contains redundant information. It is also possible to learn multiple predicates in CLAUDIEN. It has been applied on problems such finite element mesh design [48] and functional dependency discovery [92].

FOIL [127] is a top-down ILP-based concept discover system. It starts with an overly general hypothesis and refines it by adding a literal to the body at a time. Best refinements are chosen based on the information gain metric. FOIL requires mode declarations as well. It is capable of learning recursive concept descriptors. Although FOIL requires negative target instances, they need not necessarily be explicatively provided. FOIL can infer the negative target instances based on the Closed World Assumption [131]. FOIL has been applied on problems such as diterpene structure elucidation [56] and mutagenicityin nitroaromatic compounds [117].

PROGOL [106] is a top-down ILP-based concept discovery system. It performs a search on refinement graph which is bounded by a bottom clause. PROGOL needs mode declarations and negative target instances. PROGOL employs inverse entailment during the refinement step. Clauses are evaluated based on their *f-value*. It is capable of learning recursive rules. PROGOL has been applied on real world problems such as the yeast production process [50], discovery of protein folding signatures problem [160], traffic problem [55], and part-of-speech disambiguation [8].

ALEPH [150] is very similar to PROGOL in terms of search strategy and refinement operator but employs more sophisticated evaluation metrics such as pbayes [42] and wracc [125]. ALEPH has been applied to biochimical data to find hexose binding sites of protein-sugar models [116], word sense disambiguation [148].

2.1.2 Concept Rule Induction System (CRIS)

CRIS an ILP-based predictive concept discovery systems that employs association rule mining techniques [44] to induce strong and frequent concept descriptors.

CRIS inputs target instances, background data and outputs concept descriptors in the form of Horn clauses. CRIS directly works on relational data which contains only positive target instances. It generates the negative target instances based on the Closed World Assumption whenever necessary. CRIS is a parametric system as it inputs minimum support and minimum confidence values to evaluate the concept descriptors, and maximum concept descriptor length to limit the search space. Different than many other ILP-based concept discovery systems it does not require mode descriptions, which may require strong expertise to be defined, but instead utilizes referential constraints, support and confidence values to prune the search space. CRIS is able to induce recursive concept descriptors, in such problems the target instance set is also treated as background data.

Figure 2.1 describes the flow of CRIS. It is composed of four main components.

1. **Generalization:** In this step the most general two literal concept descriptors, head literal being the target relation and the body literal being a relation from the background relations, are generated. For this purpose CRIS examines each argument of every predicate if it should participate as a variable argument or a constant argument during the concept induction process. A nominal argument participates as a constant in the concept induction process if it appears at least $min\_sup \times number\_of\_uncovered\_target\_instances$ times in the table, otherwise it is represented by a variable argument. If an argument is a numeric value, instead of representing it as a constant it is represented through ranges. To find the feasible ranges, the column is sorted in an ascending order, partitioned into slots and lower bounding values of the slots are assigned as representative values.

2. **Specialization:** In this step concept descriptors of length l are refined to build the concept

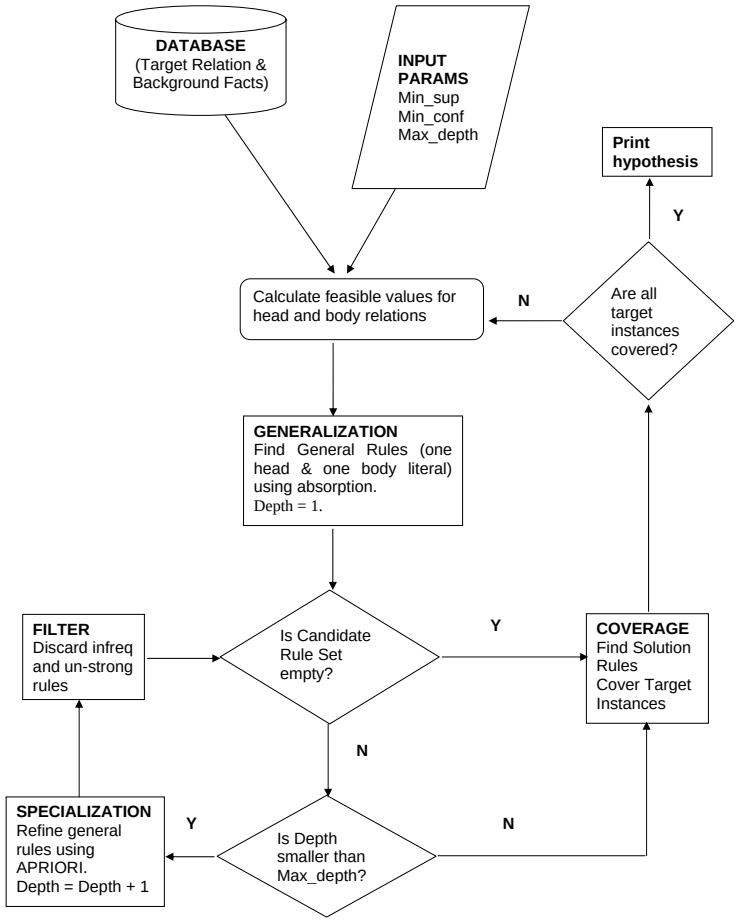


Figure 2.1: Flow Chart of CRIS

descriptors of length *l+1*. For this purpose CRIS compares each concept descriptor to every other concept descriptor and unifies the ones that differ by exactly one body literal. CRIS utilizes Apriori-based specialization operator [12]. A further specialization step is employed that unifies the existential variables of the same type in the body of the concept descriptor. By this way, concept descriptors with relations indirectly bound to the head predicate can be captured.

3. **Filtering:** This is a two fold step. In the first step goodness values of the concept descriptors are calculated, and later in the second step the concept descriptors are either pruned, refined one more step or added into the final solution set.

   Goodness of a concept descriptor is determined by its support and confidence value. Support of a concept descriptor is the ratio of number of positive target instances it captures over number of target instances. Confidence of a concept descriptor is the ratio of number of positive target instances it captures over number of instances that are deducible by its body literals. To calculate these values concept descriptors are translated into SQL queries and are run against the input data.

   CRIS employs a two phase pruning mechanism. In the first phase concept descriptors that violate the referential constraints are pruned. The remaining concept descriptors are pruned based on

their support and confidence values:

1. If a concept descriptor has a support value less then minimum support it is pruned.
2. If a concept descriptor has a support higher than minimum support but a confidence value less than minimum confidence, it is pruned one more step.
3. If a concept descriptor of length larger than 2 has a confidence value less then either of its parents it is pruned.
4. If a concept descriptor satisfies both the minimum support and minimum confidence it is added into the final solution set. If more than one concept descriptor is added into the solution set in this step the best concept descriptor is chosen based on the f-metric [71].

4. **Coverage:** In this target instances explained by the best concept descriptor found in the previous are removed from the target instance set.

Support of a concept descriptor is the ratio of the number of positive target instances captured by the rule over number of target instances.

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

Confidence of a concept descriptor 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.

$$\text{confidence}(h \leftarrow b) = \frac{|\text{bindings of variables for h that satisfy } h \leftarrow b|}{|\text{bindings of variables for h that satisfy } b|}$$

In Table 2.2 and Table 2.3, respectively, we provide support and confidence queries for the concept descriptor

elti(A, B):-husband(C,A), brother(C,D)

Table2.2: Confidence Query

Nominator	*SELECT COUNT(CONCAT(r1.name2,'-',r3.name)* *FROM elti r0, husband r1, brother r2, person r3* *WHERE r1.name2=r0.name1 AND r3.name=r0.name2 AND r1.name1=r2.name1*
Denominator	*SELECT COUNT(CONCAT(r1.name2,'-',r3.name)* *FROM husband r1, brother r2, person r3 WHERE r1.name1=r2.name1*

Table2.3: Support Query

Nominator	*SELECT COUNT(CONCAT(r1.name2,'-',r3.name)* *FROM husband r1, brother r2, person r3 WHERE r1.name1=r2.name1*
Denominator	*SELECT COUNT CONCAT(r0.name1,'-',r0.name2)* *FROM husband r1, brother r2, elti r0* *WHERE r0.is_covered = 0 AND r1.name1=r2.name1 AND r1.name2=r0.name1*

If the data set contains noisy data, CRIS generally needs to loop through these four steps multiple times until the remaining number of unexplained target instances drops below a certain threshold or no concept descriptors can be found that explain the remaining target instances.

2.2 Scalability and Efficiency of ILP-based Concept Discovery Systems

ILP-based concept discovery systems usually build large search spaces mainly due to the refinement operators. Evaluation of these search spaces is a computationally dense task due to the following reasons:

1. Evaluation of the queries requires as many table joins as the length of the concept descriptors and in some cases range queries are executed,
2. Such queries are run against large data sets.

To improve the evaluation several studies based on techniques such as parallelization, memiozation, query optimization, search space sampling, data set sampling have been proposed with promising experimental results.

In this section we firstly provide related work on memoization and parallelization to improve ILP-based concept discovery systems. Later, in Section 2.2.3, we summarize other approaches to improve search space evaluation process of such systems.

2.2.1 Related work on Memoization in ILP-based Concept Discovery Systems

As mentioned in the previous section ILP-based concept discovery systems usually suffer from efficiency and scalability issues due to the long running search space evaluation step. One of the solutions proposed for this problem is incorporating memoization to ILP-based concept discovery systems. Memoization is a commonly employed technique to improve running time of a computer program without modifying its internals [73]. It has widely been employed in query optimization [118, 101], model checking [51], parsing [124], and compiler design [72].

Memoization is useful in ILP-based concept discovery systems as such systems usually execute repetitive evaluation queries, some evaluation queries contain some others as sub-queries or some queries have common sub-queries.

Query Packs [21] is a memoization based technique proposed for handling queries that share a common sub-structure. In ILP-based concept discovery systems the search space is generally represented as a search lattice where hypotheses close to each other in the lattice are also similar in the concept they represent. Query Packs approach proposes to reorganize the search lattice in such a way that common parts of the such hypotheses are executed once, and those partial results are used to complete the evaluation of the hypotheses. Query Packs approach works for hypotheses that are of the same in length, i. e. for the concept descriptors represented at the same length. The Query Packs approach is implemented as en extension to standard Prolog engine.

Common Prefixes [137] makes use of iterative hypothesis construction property of ILP-based concept discovery systems and uses memoization to improve the search space evaluation step. It assumes that

if the partial hypothesis C is good in quality, i.e. explains many positive target instances and few negative target instances, then its refinement will be generated. Common Prefixes saves the result of C in order to calculate the coverage of its refinement.

The Query Flocks approach [159] is proposed to handle hypotheses that differ only by the constants they contain in the *where* clause and are exactly the same in all other aspects. In such cases, the Query Flocks approach suggests to run a query that performs a full join on the relations that constitute the *from* clause and select the desirable query results, i. e. tuples that relate to the constants in the original query.

Coverage Caching [8] is an other memoization based technique to improve search space evaluation step of ILP-based concept discovery systems. Similar to Query Packs and Common Prefixes it relies on the refinement by one literal at a time and generality ordering property on the concept descriptors based on θ-subsumption. Different than Query Packs and Common Prefixes it stores the non-promising hypotheses instead of promising ones. Any hypothesis that contains a previously stored non-promising hypothesis as a substructure is pruned without being evaluated. This method has also been developed for and implemented as an extension to Prolog engine.

In the Incomplete Tabling approach [136] hypotheses are evaluated until certain number of target instances are found that make them good enough to be considered in the later iterations and those target instances are stored in a look-up table. Once a hypothesis is generated that contains a subgoal that is stored in the look-up table, firstly the refined hypothesis is tested for goodness against the target instances that satisfy its subgoal. If an evaluation of a hypothesis based on the saved target instances is enough to consider it as good, it is processed in the next level without further evaluation. If, on the other hand, the saved target instances fail to consider it as a good hypothesis, it is tested against the entire data set and if it succeeds its solution set is inserted into the look-up table.

ILP-based concept discovery systems that make use of memoization, and ILP itself, has the potential of facing memory overflow problems [25]. Such encounters have already been reported in [137] and [158]. In order to cope with such situations a number of methods and customized data structures have been proposed.

To deal with memory problems, Query Packs reorganizes the evaluation of the hypotheses that form the search space in such a way that a query result is used shortly after it is tabled and then removed afterwards from the look-up table. In the Incomplete Tabling approach when a memory problem is detected, the least recently entry is removed from the look-up table. Sagonas and Stuckey [141] propose a method called *just enough tabling* which offers mechanisms to suspend and resume tabled evaluation without requiring any re-computations.

AD-trees [103] and RL-trees [60] are two special purpose data structures proposed to deal with memory requirements of ILP-based concept discovery systems that incorporate memoization. These data sets are generally employed to store full joins of multiple tables or statistics that describe the data base. Full joins over multiple tables or statistics, such as counts of certain combinations over table arguments, that describe the data set generally require large space to store. Such data structures provide memory efficient representations of such large data by for example storing combinations of table arguments that appear more than some threshold.

General purpose data structures such as tries [64] and interval lists have also been successfully employed in ILP-based systems in order to deal with memory issues. Tries are proposed to store hypotheses, while interval list are proposed to store results of the evaluation queries. While utilizing tries in ILP [32], common prefixes of the hypotheses are stored only once so providing memory utilization in

storing the search space. In order interval list be applicable in ILP-based concept discovery systems, the target instances and the background facts need to be uniquely indexed by integer values. In interval list, instead of explicitly enumerating the facts that satisfy a certain evaluation query, lower and upper bounds of the fact indexes are given if they form a sequence in terms of their indexes. Approaches such as Aleph and IndLog [31] utilize this structure to save the explained target instances.

2.2.2 Related work on Parallelization in ILP-based Concept Discovery Systems

Computational power of single processor computers is far away from meeting requirements of today's complex problems that work on huge amount of data. This has led to the development of parallel systems. In such systems, the problem or the data are divided into smaller pieces, each one is executed on a separate computing unit, and finally partial results are gathered to build the global solution.

A widely accepted taxonomy to classify the parallel systems is proposed by Flyyn [59]. This classification is based on the notations of instruction streams and data streams and classifies the parallel systems as follows:

- Single Instruction Single Data (SISD): This category represents class of machines which have a single processor which operates a single instruction stream stored in a single memory.
- Single Instruction Multiple Data (SIMD): Systems in this category consists of several processors that execute the same instruction stream on data stored at multiple memory locations. In such systems the computing units are directed by common control unit which issues the instructions to be executed.
- Multiple Instruction Single Data (MISD): In systems that belong to this class, multiple computational units perform some, not necessarily identical, instructions on a data stored in a single memory.
- Multiple Instruction Multiple Data (MIMD): This class of systems consist of multiple processing units that perform some operations on some data stored at multiple locations asynchronously and independently.

This classification has been extended by Johnson [80] to classify the systems that fall under the MIMD category. Johnson's proposal further divides such systems with respect to their memory structure: *shared memory systems* and *distributed memory systems*; and communication mechanism as *shared variables* and *message passing systems*. In shared memory systems the processors are connected to a globally available memory. In such systems usually the operating system is responsible for memory coherence. In distributed memory architectures each processor has it own memory and none is aware of the memory content of the others.

In shared memory architectures, processors are generally connected to the memory through central bus. This type of architectures usually have a scalability problem to the number of processors. To overcome such problems, hierarchical architectures are proposed where the processors located on the same board can access the memory directly, while processors placed on different boards have a controlled access to each others' memory.

In distributed memory architectures, it is not feasible to connect processor to each other via a common bus. So such systems employ message passing messages for communication purposes. Such systems

usually scale well with the number of processors. A number of message passing interfaces have been developed, Open MPI [6], openMP [7], Microsoft Message Passing Interface (MS MPI) [10] just to mention a few.

Parallelization of a serial code is achieved either by data parallelization or task parallelization. In data parallel approaches processors access different parts of the data and execute the same set of operations on it. In task parallel approaches different set of instructions are executed on data. These instructions need not necessarily access the same data at the same time but may execute instructions on different parts of the data. Task parallelization generally requires synchronization at some point in the execution.

Even though parallelization provides speedup over sequential codes, it has its own costs. Above all, parallelization of a sequential code is not always an easy task, and not all serial codes can fully be parallelized. Parallelization of a serial code requires what is called dependency analysis both on the instructions executed in the serial code, i.e. *task dependency* analysis, and the data processed, i.e. *data dependency* analysis. Then based on the dependencies the serial code goes through *decomposition* process where the problem is divided into small parts, called *task*. The number of task defined and their size define the granularity of the parallel code. A parallel code which has large number of task each of which are small in size is called *fine-grained*, while a parallel code is called *coarse-grained* if tasks are small in number and large in size. An indicator to determine the granularity of a parallel code is its *degree of concurrency* which determines the number of tasks running in parallel. Two measured derived from this indicator are *maximum degree of concurrency* and *average number of concurrency*. The former one refers to the maximum number of parallel tasks running in parallel at any instant during the execution of the parallel code, and the later one refers to the average number of tasks that run concurrently throughout the total execution of the parallel code.

Another important concern in parallelization is the so called *load balancing* issue. Decomposing the serial code into optimal number and size of tasks in not enough on its own to achieve the maximum speedup as such tasks may have varying running times on the data. Such variations in the running time may cause some processors be idle while others still perform some computations. On the other hand, even if all tasks are finished with processors minimal idle time may be engaged in heavy communication to transfer the intermediate results among each other. Several data balancing algorithms have been proposed including static mappings [30] and dynamic mappings [146].

Nevertheless, such systems hardly linearly scale with the number of processors in the systems [15], and the maximal speedup is bounded by an upper limit. Given enough number of processors, the upper limit on the speedup a parallel version of program can achieve over its sequential version is determined by Amdahl's Law [17]. The law states that if P is the proportion of the serial code that is parallelized, and *(1-P)* is the proportion that can not be parallelized hence is executed in a sequential manner, and N is the number of processors available, the maximum achievable speedup with N processors, *S(N)*, is

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

Speeding up ILP-based concept discovery, and data mining in general, through parallelization has widely been addressed in the literature. Such efforts include parallel exploration of individual hypotheses, parallel exploration of the search space, and parallel coverage.

Three parallel algorithms, namely *Count Distribution* (CD), *Data Distribution* (DD) and *Candidate Distribution* (CaD), are propsed by Agrawal and Shafer [13] to parallelize the APRIORI algorithm.

The algorithms are designed to work on distributed memory architecture and the data is evenly distributed among computers' disk beforehand. In CD each processor stores the complete strong $k - itemsets$ and redundancy comes while creating candidate $(k + 1) - itemsets$. Processors exchange local candidate $(k + 1) - itemsets$ counts to calculate the global counts. The DD algorithm utilizes the system's total memory usage by each processor counting a mutually exclusive subset of candidate $(k + 1) - itemsets$. DD has high communication cost due to exchange of local datasets. The CaD algorithm aims to build an asynchronous system by intelligently dividing the candidate itemsets and the data set among the processors in a such way that they do not require any data exchange for calculating count values. Han et al. [74] proposes *Intelligent Data Distribution* (IDD) and *Hybrid Distribution* (HD) algorithms to improve shortcomings of CD and DD, respectively. IDD focuses on improving the communication schema and load balancing problems of DD. It replaces the all-to-all communication schema of DD with point-to-point communication of processors, and improves load balancing problem of DD by sending itemsets with specific prefixes to specific processors. Communication becomes a bottleneck both for DD and IDD as more processors are added into the system or as the number of candidate $k - itemsets$ decreases. HD aims to overcome this problem by forming equal sized groups of processors, running IDD within each group to find the local candidate $k - itemset$ counts and finally running CD to find the global $k - itemset$ counts.

Another work on parallelization of association rule mining is PDM [122], which is similar to the CD algorithm in nature. While CD exchanges all itemsets, PDM exchanges local frequent itemsets with count value equal or larger than $\lfloor \frac{s}{n_p} \rfloor$, where n_p is the number of nodes and s is the minimum support value. R. Kufrin [90] proposes parallelization methods for the most computation demanding parts of the rule generator component of C4.5. Test results show that parallelization has improved running time by at least 80% over sequential execution time on 8 processors.

Parallel version of CLAUDIEN is presented in [45]. The proposed system is based on shared memory architecture and explores the search space in a parallel manner. Each processor stores the entire background knowledge while positive examples and the language bias are split among processors. Each processor executes an ILP learner on its local data and the final hypothesis is formed by combining the partial hypotheses.

Matsui et al. [98] proposes three parallelization techniques for FOIL based on splitting the search space *SSP-FOIL* algorithm, splitting the training set *TSP-FOIL* algorithm, and splitting the background knowledge *BKP-FOIL* algorithm. All algorithms work in a distributed memory environment and employ master-worker architecture. In SSP-FOIL, each processor stores the entire training set and the background knowledge while a subset of the search space. Each processor extends its local search space, i.e. adding new variables, and sends them to the master for overall evaluation. In TSP-FOIL, each processor stores the entire hypothesis space, the background knowledge and a subset of the training examples. Each processor extends its local training set through substituting the newly induced variables by the constants of the training set. Master collects the new training set items from the workers and chooses the best variables. In BKP-FOIL, the process is the same of the TSP-FOIL algorithm. A processor extends the local training set according to the background knowledge it has.

In [62], a pipelined data-parallel algorithm for ILP is proposed. The proposed algorithm $P^2 - MDIE$, parallelizes the well known ILP technique Mode Directed Inverse Entailment (MDIE) [106] by splitting examples among p processors and starting p searches in parallel. Learned rules are broadcast to all processors and tested against their portion of data and rules that evaluate *good* on all subsets are added into the final solution set.

Parallel hypothesis search technique for ILP system PROGOL is presented in [119]. They adopt gen-

erality function, *g(h)*, as the objective function and extend the notation of compression measure to $f(h) = (1 - \lambda)g(h) - \lambda n(h)$ to select which hypothesis to expand. Parallel execution of search is achieved by parallel exploration of independent hypotheses, parallel hypothesis branching, and parallel counting of *g(h)* and *f(h)*.

Data-parallel version of ALEPH is presented in [86]. They adopt master - worker architecture where master is responsible for building the bottom candidate clause and workers are responsible for evaluating the candidate clause against their local portion of data. Upon receiving answers from workers, master executes clause evaluation loop and proceeds with the next example.

Fonseca et al. [63] reports three parallel ILP systems based on different parallelization strategies. All algorithms proposed in the study employ master-worker architecture and distributed memory environment. In Parallel Coverage Test, the *pct* algorithm, positive and negative training examples are evenly split among workers. Each worker receives some hypothesis from the master and evaluates them against on its local data set. Master collects the partial scores and sums them up to calculate the final score. In Data Parallel Rule Learner, the *dplr* algorithm, positive examples are evenly divided among processors while all processors store the entire set of negative examples. All processors start learning rules according to their data sets, exchange these rules among each other to build the global coverage values. Each processor removes the examples from its local data set covered by the best rule. In Data Parallel ILP, the *dpilp* algorithm, positive and negative examples are divided among processors. Each processor executes an ILP learning system on its local data set and found rules are combined to form the global hypothesis.

2.2.3 Other Approaches

Besides memoization and parallelization considerable effort has been conducted on improving the scalability of ILP-based concept discovery systems in terms of query optimization, hypotheses sampling, data sampling, and lazy evaluation. In this subsection we present such studies.

One of the efforts to improve the search space evaluation of ILP-based concept discovery systems in term of query optimization is Query Transformations [41, 40]. The Query Transformations approach does not change the size of the search space, but transforms the evaluation queries in such a way that they are more efficiently evaluated. The proposed transformations are implemented on Prolog engine. The first transformation is Theta-transformation, t_θ, is achieved by removing the redundant literals from the goal set. In the second transformation called Cut-transformation, $t_!$, goals that substitute the body of the clause are split into groups such that solution of a group does not alter the solution of another. In Prolog manner failure in the evaluation of some goal requires reevaluation of the goals that precede it in the order. Placing the clauses that are independent from each other into different groups and evaluating the groups in a sequential manner prevents the reevaluation of some goals that will not affect the solution of failing goal. The third transformation is Once-transformation, t_o further divides the classes formed by $t_!$ by collecting prior information about the groundings of variables. The last transformation defined is Smartcall-transformation, t_s, makes use generality ordering of concept descriptors. It stores the solutions of the parent clauses to evaluate their refinements and only runs the refined concept descriptors for the newly added variables.

Another effort in improving the search space evaluation step by means of query optimization is Reordering of Literals [154]. In this study literals that form the body of concept descriptor are ordered based on their selectivity. Evaluation of the search space is achieved by reordering the literals in such a way that a more selective literal precedes the less selective one in the body of the concept descriptor.

Another approach to improve running time of ILP-based concept discovery systems is minimizing the number of hypotheses to be evaluated. One of the approaches [32] to realize this is based on collecting statistical information about data set, such as the number of the occurrences of constants in data set and perform early pruning based on these values. A second commonly employed method to prune the concept descriptors without evaluating them is achieved by introduction of mode declarations [156]. Mode declarations put constraints on what kind of arguments can appear in what position is the literals. Although such declarations reduce the number of concept descriptors in a great deal, they are generally hard to be formulated and may require expertise knowledge. A similar approach for early pruning is employed by CRIS as well. In CRIS concept descriptors are pruned without being evaluated if they do not satisfy the data integrity constraints. Compared to mode declarations, pruning based on data integrity is easier to implement as such knowledge is already available with the relational schema.

Another direction to minimize the number of concept descriptors to be evaluated is based on search space approximation. DiMaio and Shavlik [47] propose a neural network based method to approximate the search space. The neural network consist of one fully connected hidden layer. In the proposed approach the authors define C as the saturated clause that is the bottom clause and C' as the variablezed form of C. Inputs of the neural network are $\vec{x}$, $\vec{y}$; and the number of literals in C, the number of distinct variables in C', number of distinct variables that appears more than once in C', the average number of times each variable appears in C', and the length of the longest variable chain appearing in C'. $\vec{x} = \{x_1, x_2, \ldots, x_{\perp_i}\}$ is defined as follows:

$$x_k = \begin{cases} 1 & \text{if literal k is chosen in construction of } C \\ 0 & \text{otherwise} \end{cases}$$

Definition of $\vec{y}$ is similar to of $\vec{x}$, y_k is 1 if C' models the target instance, 0 otherwise.

Another study to reduce the search space size is proposed by Srinivasan and Kothari [153]. In their study, the authors propose a dimensionality reduction procedure to remove the statistically irrelevant literals from the hypotheses each of which are derived from a bottom clause, C. To represent a hypothesis, C' as a vector, a literal that exists in C but is missing in C' is assigned value 0, and 1 otherwise. The authors employ Principal Component Analysis [11] and RReliefF [135] as dimension reduction algorithms.

Evaluation of the concept descriptors is costly not only because the queries are complex and are larger in number, but also because they are tested against large data sets. Methods such as data sampling [151], have been proposed to cope with the large amount of data. *Subsampling* and *Logical Windowing* are two data sampling methods proposed by Srinivasan [151]. In Subsampling, a user defined number of instances at random and with replacement are selected from the target instance set, $E_i \in E = E^+ \cup E^-$. At each iteration E_i is updated by removing the explained target instances by the clauses found in the previous iteration. In the Logical Windowing induces theories are induced and tested on different sample data sets, respectively, E_i and $Train_i$. $Train_i$ is updated by appending the misclassified target instances in E_i at the end of each iteration, and E_i is updated by removing the correctly modeled target instances.

Other methods proposed to improve ILP-based concept discovery systems include employing genetic algorithms [19, 76], stochastic matching [143], lazy evaluation [32], different representation settings [155], and memory-wise scalability methods [24].

2.3 Related work on Graph-based Concept Discovery

Logic based approaches have long dominated the area of multi relational data mining. Another arising competitor in concept discovery is based on representing the data on graph and employing graph mining algorithms.

Graphs are one of the fundamental data structures and representation frameworks of computer science. A graph, G, consists of a non-empty set of *vertices* or *nodes*, V, and a set of *edge* or *arcs*, E. Edges connect 2 element subsets of nodes, which are called *endpoints* of the edges [34]. There also exist graphs with empty set of vertices, yet no conclusion of official acceptance of such structures is reached [75].

Formally graph is defined as

$$G(V,E,f) = \begin{cases} V & \text{a set of vertices} \\ E & \text{a set edges connecting some pairs of vertices in V} \\ f & \text{a mapping function } f : E \rightarrow V \times V \end{cases}$$

Graphs appear as one of the best structures to represent structured data [161] and graph based data mining has attracted great attention in recent years. Although there exist many different graph data mining algorithms, they rely on certain theoretical aspects of graphs. Those can be listed as follows:

- Supgraph categories: Algorithms that aim to find substructures within a graph may limit themselves with certain types of substructures. An algorithm may look for subgraphs for type of induced graphs, connected subgraphs or paths.
- Supgraph isomorphism: Algorithms that look for similarities among graph structures needs somewhat to verify that two structures are identical in some sense. Subgraph isomorphism provides background for such justifications by providing mappings among the vertices and edges of the subgraphs.
- Graph invariants: Graph invariant are numerical properties that describe a subgraph in substructure-wise, such as the number of incoming edges of a vertex but not the label of the edges. They are widely used for by means of pruning criteria in search of isomorphic structures in a graph. While such properties are not enough to claim that two substructures are isomorphic they are enough to claim that they are not.
- Mining measures: Finding frequent substructures is not enough on its own is not enough on judging whether the found pattern is good enough. Such a frequently appearing substructure may be overly general or overly specific. This category deals with such goodness measures which are more or less inherited from the machine learning domain.
- Solution methods: This category deals with the searching techniques on graph data. Such searches may be complete, heuristic, kernel function based, ILP-based, and inductive database approaches.

Graph based data mining has been employed in varying domains which include from bioinformatics binding site extraction of protein molecular surfaces [91, 18], from engineering domain operations research formulations [138], from WWW domain social network analysis for business purposes [27].

Top-down ILP-based concept discovery systems are suspectable to the so called local maxima or local plateau problems [128, 104]. The local plateau problem is a certain state in the hypothesis space where traditional refinement operators become insufficient to improve concept descriptors' quality by refining a single body literal. Graph based approaches were first introduced by Richards and Mooney [133] in context of concept discovery to over come this problem. Mooney and Richards propose in their work to replace the refine by one literal at a time operator of ILP by refine by multiple literals that form a path among variables of the literals of the arguments. Following the promising results of this study several a number of other studies [38, 70, 164, 66] have been proposed that have get their theoretical background from graph data mining algorithms. Theoretical background providing relationship between graphs and logic are provided in [132, 162, 26, 147].

Graph-based approaches on concept discovery can broadly be classified as substructure based approaches and path-finding based approaches. The former approach relies on the idea that, if there exists a concept in the graph data, it should appear as frequent similar substructures. The later one follows a relatively similar analogy, if a concept exists in a graph data it should appear as frequent, fixed-length paths that connect literals that form the definition of the concept rule.

Other then the structure they look for, graph based concept discovery systems differ from each other by means of representation of the data, i. e. labeling the edges, search technique, i. e. depth-first search vs bidirectional search, noise handling mechanisms, and concept evaluation. In the following sections we firstly introduce the most common data representation mechanism employed by graph-based discovery system. Then present the search mechanism such systems utilize, and conclude the section with summarizing some state-of-the-art graph based concept discovery systems.

2.3.1 Related Work on Graph-based Concept Discovery

Although graph is the general framework to represent data in the graph-based concept discovery, such systems differ in the type of graphs they work on, what they represent by the vertices and the edges and their searching algorithms. In this subsection we introduce some state of the art graph-based concept discovery systems. We group such studies as path finding approaches and structure based approaches.

2.3.1.1 Path Finding-based Approaches

In path finding approaches motivation is to find some paths that connect arguments of the target instances. Such approaches usually employ basic graph traversal algorithms to search for the frequently appearing paths.

One of the earliest works that introduced graph based approaches to the concept discovery problem is by Richards and Mooney [133] called Relational Pathfinding. In this approach data is represented as an undirected graph where nodes represent relation arguments and edges connects vertices whose values form a fact. Although it is not clearly stated in the study, our understanding is that Relation Pathfinding allows the target examples be n-ary relations while limiting the background data with binary and unary relations. The system starts with creating n disjoint graphs where $G_i, 1 \leq i \leq n$ and n being the number of arguments in a target relation, each G_i consisting of a single vertex, say v_i. The system also associates a set of *endvalues* for each G_i called EV_i^l, where i donates the graph a set is associated to and l donates the number of expansion on the graph. To find the paths that connect the arguments of

the target instance, they expand each member of EV_i^l with every related fact[1] to form EV_i^{l+1}. At the end of each expansion, the newly formed EV_i^l are tested against each other for intersection. The expansion is stopped when an intersection is a non-empty set or l reaches the user or resource defined bound. An intersection of any two endvalue set means a path is formed between two target instance arguments are formed. Partial paths are merged to form the final paths. If more than one final paths are formed, due to multiple branchings, the best one is selected based on their accuracies.

A recent work on concept discovery on graphs is *Relational Paths Based Learning (RBPL)* [66]. They represent the data with a labeled undirected graph, where vertices store facts from the background data and edges connect nodes that have an argument of the same type and value in common. Edges are labeled after such shared constants. RBPL defines the following five structures that are employed in the concept induction process:

- **Structured Item Space (SIS)**: Holds the background data.
- **Expanded SIS (E-SIS)**: This is the enhanced form of the SIS after the application of the domain theories, if provided any.
- **Relational Path (RP)**: Any path in SIS or E-SIS that connects arguments of a target instance.
- **Specialized RR (S-RP)**: Refined version of RP which contains unary relations that define the constant in RP.
- **Generalized S-RP (S-RP)**: Generalized version of S-RP which results after variablizing the constants with unique variables.

To find a path that connect arguments of a target relation, RBPL randomly picks a number of target instances and constraints the source, s and target, t, nodes based of them. If t is a target instance of form *t(A, B)* then s is *t(A, *)* and the t is *p(?, B)*, where p is a relation from the background data whose second argument is B. To find the paths, RBPL employs breadth first search. RBPL requires user inputs to define the number of target instance to find concept descriptors and maximum length of the concept descriptors.

Although applicability of the domain theories is proposed in [66], they are not implemented in RBPL. In a later study [65] applicability of the domain theories are elaborated and experimentally discussed. This later study also discusses the induction of recursive concept descriptors on graph based concept discovery. To learn recursive rules, the target instances are also included into the SIS.

Both Relational Path Finding and RPBL are of the form of top-down ILP-based concept discovery. Mode Directed Path Finding [120] is an application of graph based concept discovery on saturated bottom clauses, i. e. a form of bottom-up ILP-based concept discovery. It differs from Relational Path Finding and RBPL not only by means of working on saturated bottom clauses, but also by employing hypergraph as a representation framework instead of graph. Each literal in the body of the saturated bottom clause is represented as a vertex in the hypergraph. Edges are created based on the input / output mode declarations among arguments of the literals. The head literal is represented with n distinct vertices, n being the arity of the head literal, and those vertices are source for the paths. Initially each path is of length 0, and each path is associated with a list that, $L_i, 1 \leq i \leq n$, and contains the value of the argument the vertex has. Vertices are expanded in a breadth first manner and at the end of each expansion the newly reachable variables are added into the appropriate list. To find the final concept

[1] Two facts are related if they share an argument with same type and value in common

descriptors, Mode Directed Path Finding compares each list against the every other list and if finds an intersection between two lists merges their associated paths. Mode Directed Path Finding aims to finds all possible paths that connect some head literal arguments, hence it does not stop expanding the graph once some concept descriptors are found.

Another study that works on saturated bottom clauses and makes use of mode declarations is Head Output Connected (HOC) Predicate learning. HOC class of clause is a special case of the clauses induced by general path learning systems. A definite clause $h \leftarrow b_1, ..., b_n$ is said to be HOC iff and only if

1) Input variables for each body literal are either subset of the head input variables or are found in a previous body atom, and

2) All its head output variables are instantiated in the body.

In the HOC system, the body literals of the bottom clause form the vertices of the graph. Starting with length, d, 1, it finds paths of length d that connect the output variables of the head to its input variables. For this purpose it employs breadth first search, and finds all paths of length at most *max_depth* which is a user provided parameter and put upper bound on the length of paths. Once all paths are discovered they are evaluated based on their compression and the best one is output as final solution clause.

2.3.2 Structure-based Concept Discovery

Graph based concept descriptors that employ substructure-based approaches relay on the idea that a concept should appear as frequent similar substructure with in graph representation of the data. Such systems usually start with multiple substructure what contain a single vertex or a single edge and expand them until some meaningful substructures are discovered. They usually employ some graph isomorphism algorithms to compare the formed substructures.

SUBDUE [78] is one of the pioneering works in the graph based concept discovery problem. SUBDUE inputs a labeled, directed multi graph where arguments are represented as vertices and edges connect arguments that forms a fact and are labeled after the relation name. Initially SUBDUE assumes each vertex as a valid substructure. At each iteration, each substructure is expanded by one edge in all possible ways. At the end of each expansion the discovered substructures are evaluated and pruned based on their goodness at compressing the initial graph. Their compression goodness is evaluated based on the Minimum Description Length (MDL) [134] value. SUBDUE employs inexact graph graph matching which is limited by user defined distortions. By this way, SUBDUE can handle noisy data or may find concepts descriptors for instances that have slight differences in representation. SUBDUE is similar to WARMR in a way that it does not output a concept descriptor for a specific target but hierarchies of concepts that are significant within the graph data.

SUBDUECL [69] is an extended version of SUBDUE to discover concept descriptors for a specific target relation. Basically it employs the same representation framework and core functions with SUBDUE, but differs in learning and evaluation mechanism. To evaluate the substructures, it employs set covering function instead of compression. Quality of a substructure is calculated with respect of the number of positive target instances it does not cover and negative target instances it covers. It employs constrained beam search and considers only as many substructures as the size of the beam at a time. It maintains three lists to keep the substructures, *ParentList*, and *ChildList*, *BestList*. ParentList

stores the current substructures, the ChildList stores the substructures that are resultant of expanding the structures in the ParentList, and the BestList stores a subset of substructures in the ChildList that model at least one positive target instance, each one limited with item count equal to the beam size. If after an expansion no substructure is added to the BestList, the beam size is increased to test the other structures. Initially the ParentList is built up adding all substructures with one vertex, the ChildList is built up by expanding these initial substructures in the ParentList by one edge in all possible ways.

Graph-Based Induction (GBI) [164] and Concept Learning From Inference Patterns (CLIP) are two concept learning system based on substructure discovery. The former one embodies the concept induction process of the later one. CLIP relies on the idea that important patterns in the data should appear as frequently repeating substructures within the graph representation of the data. CLIP utilizes colored directed graph to represent data where each vertex is associated with an edge and several colors. Colors, in this representation, encode attribute values of a fact the vertex is associated with. CLIP employs beam search to find patterns and at each iteration compresses the graph with replacing the important substructures with e new vertex. CLIP considers each pair of connected vertices, called *patten*, as promising substructure. CLIP maintains a lists called *view* each of which contain typical, i.e. similar, patterns. To check the similarity of patterns, different than SUBDUE, it employ graph identity instead of graph isomorphism. To form concept descriptors, CLIP follows a three step procedure. In the first step, called *pattern modification*, CLIP examines each view in turn. Firstly, it picks each pattern in the currently analyzed view and replaces its occurrences in the input graph with a new vertex. Then adds every connected vertex pairs into the view and new patterns. In the second step, called *pattern combination*, patterns in each updated view are combined to form new view. To combine views, all possible combinations of patterns are considered. In last step, called *pattern selection*, best patterns, determined by degree of reducing the graph when replaces by a new vertex, are selected from each view. In CLIP Quality of a substructure is determined by the number of vertices in the substructure and the number of different colors attached to the vertices.

GBI basically discusses how different types of problems can be represented as colored directed graphs. This study is further extended by Matsuda et al. [97] to handle self loops and incorporating new selection criteria.

CHAPTER 3

MEMOIZATION TO SCALE UP ILP-BASED CONCEPT DISCOVERY SYSTEMS

Although ILP-based concept discovery systems have applications in a wide range of domains such as bioinformatics [95, 160], biochemistry [116, 50], engineering [48, 58], and enviromental sciences [22, 55], they still demonstrate scalability and efficiency problems. These deficiencies are generally due to the evaluation of the large search spaces such systems create. Costa et. al. [41] reported that ALEPH spends almost 90% of its total execution time on evaluating the search space when run on the Muta data set, and around 80% of its total execution time on the PTE data set. Similar results are reported for WARMR by Blockeel et. al. [21].

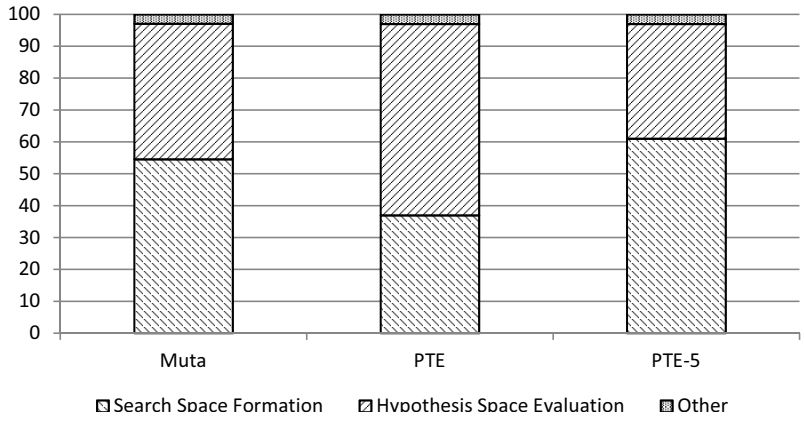


Figure 3.1: Relative running times of components of CRIS

In Figure 3.1 running times for the major components of CRIS are plotted for the Muta, PTE, and PTE-5 data sets. CRIS also has a long running time for the search space evaluation step as well as for the search space construction step. These results suggest that improving running time of ILP-based concept discovery systems is closely related with the improvement in the search space evaluation process, and CRIS also requires improvements in the search space construction step.

In the following subsections, we propose three techniques to improve running time of ILP-based concept discovery systems. These techniques are based on memoization and are basically about storing

some information in look-up tables in order to avoid calculations of already calculated functions or to improve running time of certain functions. Although the problems these techniques address are common to many ILP-based concept discovery systems, in this work, we discuss them with reference to CRIS and implement them as extensions to CRIS.

The first technique, called Tabular CRIS [112], improves running time of CRIS by targeting the search space formation and evaluation steps of CRIS. Tabular CRIS modifies the removing repeating concept descriptors function of the search space formation step, and query evaluation mechanism of the search space evaluation step.

The second technique, called Tabular CRIS-wEF [113, 115], modifies the structure of the concept evaluation queries and the Covering function in such a way that if a concept descriptor evaluation query is executed in a prior epoch, its result can still be retrieved from a look-up table.

The third technique, called Selective Tabular CRIS-wEF [114], aims to improve the memory requirement of Tabular CRIS-wEF by defining policies on what and how long to store in the look-up tables.

In the following subsections we discuss these methods in detail, and elaborate their applicability to ILP-based concept discovery systems other than CRIS.

3.1 Tabular CRIS

In ILP-based concept discovery systems, concept descriptors are refined by one literal at a time. Based on the search direction, a concept descriptor is refined either via a generalization operator or a specialization operator. Although all concept descriptors of length *l* are distinct, their refinements may be identical. Such cases are due to:

1. Different renamings of the arguments of the same type,
2. A concept descriptor of length *l* may share different literal subsets of size *l-2* in the body with more than one concept descriptor.

In Table 3.1 and Table 3.2 we demonstrate examples for each case. In order to minimize the redundant computations, such repeating concept descriptors need to be removed from the search space.

Table3.1: Concept descriptors with different renamings

C1	elti(A,B):-wife(C,D), mother(A,C)
C2	elti(A,B):-mother(A,C), daughter(D,A)
C3	elti(A,B):-mother(A,C), daughter(C,A)
C1 ∪ C2	elti(A,B):-wife(C,D), mother(A,C), daughter(D,A)
C1 ∪ C3	

In the original implementation of CRIS, firstly all concept descriptors are refined in all possible ways, then a linear search [85] is employed to remove the repeating concept descriptors.

In order to improve the running time of the search space construction step of CRIS, we propose to replace the linear search with a hash based search. Once a concept descriptor is refined, before it is

Table3.2: Concept descriptors sharing different subsets of literals

C1	elti(A,B):-wife(A,C), son(C,D)
C2	elti(A,B):-wife(A,C), mother(B,D)
C3	elti(A,B):-son(C,D), mother(B,D)
C1 ∪ C2	elti(A,B):-wife(A,C), son(C,D), mother(B,D)
C1 ∪ C3	

added into the search space it is searched in a look-up table. Failure in the search means that the concept descriptor is not generated before, and it is added both into the search space and into the look-up table. Hit in the search, on the other hand, means that the concept descriptor is already generated and added to the search space, so no action is taken.

This strategy has a three-fold contribution to the system:

1. Searching for an element in a hash table is a constant time operation,
2. Unnecessary growth of the search space is avoided during its construction step,
3. The extra computational cost for resizing the search space container is avoided.

Algorithm 2 and Algorithm 3 outline the original algorithm and the proposed algorithm, respectively. The function *unifiable* tests whether two concepts descriptors are unifiable, and if so the function *unify* unifies them. *push_back* and *erase* are member functions that adds an element to the end of a vector, and removes an element at the given position, respectively.

In CRIS, two concept descriptors map into the same confidence evaluation query if they differ only by the variables that are bounded to the head literal. Similarly, two concept descriptors share the same support evaluation query if they differ by literals that only have unbounded variables. Such examples are provided in Table 3.3 and Table 3.4.

Execution of the evaluation queries is one of the most costly process in ILP-based concept discovery systems. Execution of each such query requires as many table joins as the length of the concept descriptor, and in some cases range selections on numerical attributes are performed [139, 16]. In order to avoid executions of the repeating evaluation queries we propose a memoization based technique which mimics a query cache.

In the proposed approach, before an evaluation query is sent to the database engine, it is first searched in a look-up table. If the search results with a hit, which means query is executed before, its result is retrieved from the look-up table. If the search results with a miss, which means the evaluation query is not executed before, the evaluation query is sent to database engine for execution and is inserted into the look-up table along with its result. The proposed approach is outlined in Algorithm 4.

Support and confidence evaluation queries are *SELECT COUNT* type queries that return the number of the target instance a concept descriptor explains and the number of the back ground knowledge instances that the body of the concept descriptor explains, respectively. As at the end of each epoch the target instance set is altered in course of the Covering function, the same support evaluation query generated at different epochs will have different support values, while the confidence evaluation query will return its previous result. Due to this fact, support and confidence evaluation queries are stored at

distinct look-up tables and the one that stores the support evaluation queries is cleaned up at the end of each epoch.

Algorithm 2 CRIS: Search Space Construction

Requires: C_l: Vector of concept descriptors of length l

Ensures: C_{l+1}: Vector of concept descriptors of length $l+1$

```
1:  for i = 0 to C_l.size − 1 do
2:    for j = i + 1 to C_l.size do
3:      if (unifiable(C_l[i], C_l[j])) then
4:        C_{l+1}.push_back(unify(C_l[i], C_l[j]))
5:      end if
6:    end for
7:  end for
8:  for i = 0 to C_{l+1}.size − 1 do
9:    for j = i + 1 to C_{l+1}.size do
10:     if (C_{l+1}[i] == C_{l+1}[j]) then
11:       C_{l+1}[j].erase()
12:     end if
13:   end for
14: end for
```

Algorithm 3 Tabular CRIS: Search Space Construction

Require: C_l: Vector of concept descriptors of length l

Ensure: C_{l+1}:Vector of concept descriptors of length $(l+1)$

Local Variables: specHash: Look-up table, uC: Concept descriptor

```
1:  for i = 0 to C_l.size − 1 do
2:    for j = i + 1 to C_l.size do
3:      if (unifiable(C_l[i], (C_l[j])) then
4:        uC = unify(C_l[i], C_l[j])
5:        if (find(specHash, uC) = NULL) then
6:          C_{l+1}.push_back(uC)
7:          specHash.insert(uC)
8:        end if
9:      end if
10:   end for
11: end for
```

Table3.3: Concept descriptors mapping into the same confidence query

C1	elti(A,B):-husband(C,B),brother(C,D),brother(D,C)
C2	elti(A,B):-husband(C,A),brother(C,D),brother(D,C)
Query	SELECT DISTINCT CONCAT(r1.name2,'-',r4.name) AS A1 FROM husband r1,brother r2,brother r3,person r4 WHERE r1.name1=r2.name1 AND r1.name1=r3.name2 AND r2.name1=r3.name2 AND r2.name2=r3.name1

Table3.4: Concept descriptors mapping into the same support query

C1	elti(A,B):-wife(A,C),son(D,E)
C2	elti(A,B):-wife(A,C),sister(D,E)
Query	SELECT DISTINCT CONCAT(r0.name1,'-',r0.name2) AS A1 FROM wife r1,elti r0 WHERE r0.is_covered_aux = 0 AND r1.name1=r0.name1

Algorithm 4 Tabular CRIS: Search Space Evaluation

Require: C: Vector of concept descriptors of length l

Ensure: scC:Vector of strong and confident concept descriptors of length l

Local Variables: supHash: Look-up table for support queries, confHash: Look-up table for confidence queries

```
for i = 0 to C_l.size do
  supSQL = buildSupportQuery(C[i])
  it = supHash.find(supSQL)
  if (it ≠ null) then
    C[i].sup = it.value
  else
    C[i].sup = executeQuery(supSQL) supHash.insert(supQuery, C[i].sup)
  end if
  if (C[i].sup ≥ min_sup) then
    confSQL = buildConfidenceQuery(C[i])
    it = confHash.find(confSQL)
    if (it ≠ null) then
      C[i].conf = it.value
    else
      C[i].conf = executeQuery(confSQL)
      confHash.insert(confSQL, C[i].conf)
    end if
    if (C[i].conf ≥ min_min_conf) then
      solC.push_back(C[i])
    else
      scC.push_back(C[i])
    end if
  end if
end for
```

As in the search space construction step, in the search space evaluation step look-up tables are implemented as hash tables. In this implementation evaluation queries are treated as *key* values and the query results are treated as *mapped* values. In Algorithm 4, *buildSupportQuery* and *buildConfidenceQuery* functions translate the concept descriptors into corresponding support and confidence queries, respectively. The *executeQuery* function executes a query and returns its result. *it* is an iterator addressing to a particular element in the hash table and its *second* value corresponds to the query result. The *insert* and *push_back* functions are defined as in Algorithm 3.

3.2 Tabular CRIS-wEF

In presence of noisy and incomplete data, which is usually the case with ILP-based concept discovery systems [53, 20], CRIS usually needs to run multiple epochs to find the entire solution set. At each epoch such systems discover some concept descriptors with high accuracy and low coverage, and execute another epoch to discover concept descriptors that explain the remaining target instances [102].

At each new epoch CRIS generates some of the concept descriptors that have been generated in previous steps and some new concept descriptors. Tabular CRIS can retrieve confidence evaluation query results of such regenerated concept descriptors from the look-up table, while has to re-execute the support evaluation queries. In Tabular CRIS-wEF we address the problem of handling results of repeating queries that are generated at different epochs.

Table3.5: Daughter data set

Target Instances		**Background Instances**	
daughter(mary, james)	$\oplus$	father(james, mary)	father(david, linda)
daughter(patricia, robert)	$\oplus$	father(paul, susan)	father(robert, patricia)
daughter(linda, david)	$\oplus$	father(denis, walter)	mother(sandra, maria)
daughter(barbara, helen)		mother(amanda, gary)	mother(helen, barbara)
daughter(maria, sandra)		female(maria)	female(linda)
daughter(susan, paul)	$\oplus$	female(barbara)	female(mary)
		female(patricia)	

As an example consider the *daughter* data set given in Table 3.5, where the first argument of the predicate is related to its second arguments with the predicate name, i. e. *daughter(X, Y)* means *X* is daughter of *Y*, and *female(X)* means that *X* is a female.

In Figure 3.2 we visualize some of the most general concept descriptors that CRIS constructs at the first epoch. In the small box on the left we list the most general two literal concepts descriptors and arrows connect them to their corresponding evaluation queries.

Assuming that minimum support is 0.6 and minimum confidence is 1.0 Tabular CRIS will output s_1 as solution a clause at the end of the first epoch.

s_1: daughter(X, Y) :- father(Y, X), female(X)

The target instances explained s_1 are marked with $\oplus$ in Table 3.5. Because not all of the target instances are explained in the first epoch Tabular CRIS will perform a second epoch on the remaining target instances. In this new epoch Tabular CRIS will generate r_1, r_2, and r_3 as the most general two literal concept descriptors. Tabular CRIS needs to re-execute the query corresponding to r_1 as it has been removed from the hash table. In order to avoid such computations we propose the following modifications on Tabular CRIS's query evaluation and covering steps:

1. Query structure of support queries is changed from *SELECT COUNT* to regular *SELECT* statements. With this modification, support evaluation queries return a set of target instances that satisfy the concept descriptor, and hash table for support queries store $< query, resultset >$,

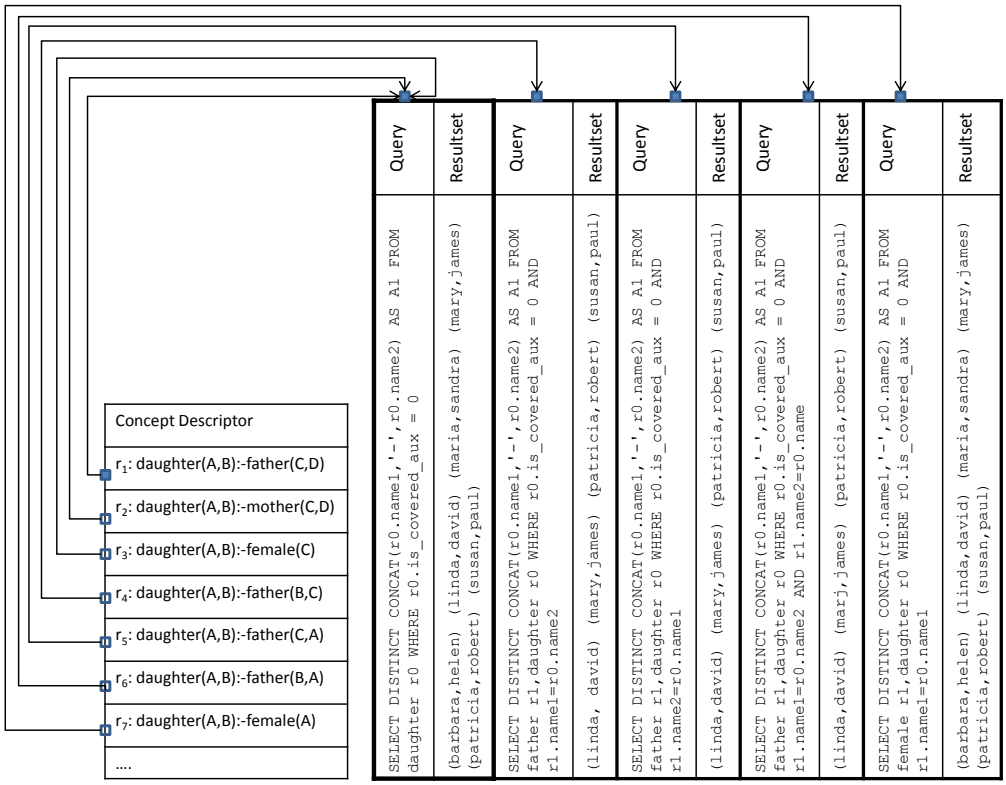


Figure 3.2: Initial state of the hash table for support queries

where query is the key and resultset is the mapped value. Confidence evaluation queries still follow the *SELECT COUNT* structure.

2. The Covering algorithm is modified in such a way that it not only removes the explained examples from the target instance set but also removes them from the hash table that stores the support queries. With this modification in the Covering algorithm, hash table for support queries stores updated result sets of the stored queries after some solution clauses are found.

In Algorithm 5 we provide the modified version of the Covering function. The inner for loop removes the explained target instances from the hash table.

Algorithm 5 Tabular CRIS-wEF: Covering Algorithm

Require: E: target instances, BF: Background knowledge, H: Hypothesis
Ensure: E: target instances not covered, Updated Memorized result sets

```
1: for all e ∈ E do
2:   if BF ∪ H ⊨ e then
3:     E = E \ e
4:     for i = 0 to i < supHash.size() do
5:       supHash[i].resultSet=supHash[i].resultSet \ e
6:     end for
7:   end if
8: end for
```

After running the modified version of the Covering algorithm, the hash table for support queries given in Figure 3.2 will be as in the Figure 3.3. With this modification on the Covering algorithm, result of the support evaluation query of r_1 can directly be retrieved from the hash table in the second epoch.

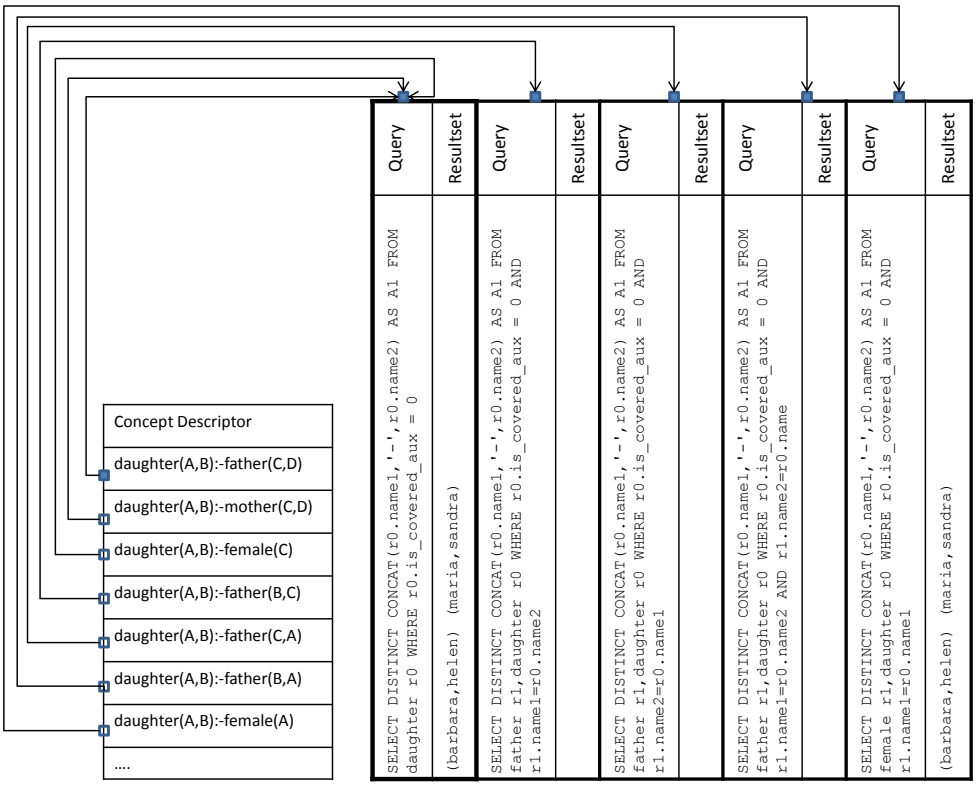


Figure 3.3: State of the support hash table after the modified covering algorithm is run.

3.3 Selective Tabular CRIS

Although Tabular CRIS-wEF improves hash table hit count of Tabular CRIS around 30% on the average, it introduces certain costs over Tabular CRIS:

1. The modified Covering function requires longer execution time.
2. Tabular CRIS-wEF requires more memory as it stores a number of tuples for each support evaluation query instead of a single numeric value,
3. Additional memory requirement, which indeed is inherit from Tabular CRIS, is for the hash tables employed in the search space construction step for removing the repeating concept descriptors.

As the experimental results show that the extra time introduced by the modified Covering function is very small, and even negligible for the experiments that have long running times, we leave off this issue.

In Selective Tabular CRIS we focus on the increased memory requirement of Tabular CRIS-wEF and propose policies to improve the memory consumption. Tabular CRIS-wEF can be considered as a greedy memoization approach as

1. It never removes a query from the hash tables even though some of them will never be generated again,
2. It never removes a refined concept descriptor from its hash table, till the end of the epoch, even though some generated refinements will never be generated again after a certain point.

During the generalization step, CRIS considers each argument of a predicate individually and decides on the values it can have during the concept induction process. To find feasible values for nominal arguments, CRIS executes the query given in (3.1).

$$\begin{aligned} &\textit{SELECT a FROM t} \\ &\textit{GROUP BY a HAVING COUNT(*) >=(min\_sup}\times\textit{num\_ucv\_ins)} \end{aligned} \tag{3.1}$$

Argument values that satisfy this query participate both as constants and as variables in the concept induction process, others are substituted with variables.

Lemma 3.3.1 *Assume that c_j^p is the set of the nominal arguments that qualify to be represented as a constant for predicate* p *at epoch j. Since the number of uncovered target instances decreases at the end of each epoch, c_j^p will be a subset of $c_k^p, k > j$, which means concept descriptors generated at epoch j will be regenerated at epoch k.*

Searching for feasible constants for numerical values is not appropriate due to the support and confidence thresholds. For this reason, feasible ranges are given through *less than* / *greater than* operators. To find the feasible range values, CRIS sorts a table in ascending order based on the numerical argument, partitions the table into slots using the equation given in (3.2), and uses the lower bordering value of each slot as a feasible numerical constant.

$$c \times k \times \frac{\#uncovered\_target\_instances}{\#target\_instances},\ 0 \leq k \leq 1.0 \tag{3.2}$$

where c is the cardinality of the table, and k is a user defined constant to determine the width of the slots.

Lemma 3.3.2 *Equation (3.2) states that width of the slots for the numerical values is determined by the number of uncovered target instances. Once some solution clauses are found, width of the slots changes, and hence the representative values, which means that an argument with a previous representative numerical value will not be generated again.*

To refine a concept descriptor, Tabular CRIS-wEF tests each concept descriptor against every other concept descriptor that succeeds it in lexicographical order and unifies the pairs that differ by exactly one body literal. If many concept descriptors succeed to pass the *min_sup* threshold, employing hash based search in the specialization step may introduce memory problems.

Lemma 3.3.3 *Let Z be the set of all possible generalizations of the predicates, and $p, q, r \in Z$; $q \neq p$; and r succeeds p and q in lexicographical order. Suppose that c_p, c_q, and c_r are three concept descriptors of length l, $l > 1$, that has p, q, and r as a predicate in their bodies. Any refinements of $\{c_p, c_r\}$ will differ from the refinements of $\{c_q, c_r\}$ with a set a predicates that contain p.*

Based on the Lemmas defined above we propose the following four policies to improve the memory consumption of Tabular CRIS-wEF.
Policy 1: A concept descriptor that contains only variable arguments should permanently be kept in the hash table.
Policy 2: A concept descriptor that contains variable arguments and nominal constant arguments should permanently be kept in the hash table.
Policy 3: A concept descriptor that contains a numerical constant should be kept in the look-up table until the end of the epoch.
Policy 4: Refinements of a concept descriptor should be kept in a look-up table until it is tested for every other concept descriptor.

Policy 3 changes the greedy memoization behavior of the search space evaluation step of Tabular CRIS wEF to selective behavior, and Policy 4 improves the memory consumption of the search space construction step of Tabular CRIS-wEF.

Algorithm 6 outlines the algorithm for search space evaluation under selective memoization. The *containsNumConst* function checks if the concept descriptor has literal with a numerical constant argument. Based on the return value of this function, queries and their result sets are stored and searched either in temporary hash tables, namely *sHashT* and *cHashT*, or permanent hash tables, namely *sHash* and *cHash*. The *buildSupportQuery*, *buildConfidenceQuery*, *executeQuery*, *find*, *insert* functions are as defined for Algorithm 4.

sHash, *cHash*, *sHashT*, *cHashT* are globally defined hash tables to store queries. Support queries that comply with Policy 1 and Policy 2 are stored in the *sHash*, the ones that comply with Policy 3 are stored in the *sHashT*. Similarly, confidence queries that contain numerical constant argument are stored in the *cHashT* table, and the others are stored in the *cHash* table. The temporary hash tables are cleaned once some solution clauses are found.

Algorithm 7 outlines the modified specialization function. The *unifiable*, *unify*, *find*, *inset*, and *push_back* functions are as defined in Algorithm 3. The *clear* functions cleans the content of a hash table.

Algorithm 6 S-Tabular CRIS: Evaluation

Requires: C_l: Vector of concept descriptors of length l

Ensures: sC: Vector of concept descriptors of length $l+1$

```
1:  for i = 0 to C.size − 1 do
2:    flag = 0
3:    q=buildSupportQuery(C[i])
4:    if containsNumConst(C[i]) then
5:      if sHashT.find(q) = NULL then
6:        sup=executeQuery(q)
7:        sHashT.insert(q, sup)
8:      else
9:        sup=find(sHashT, q)
10:     end if
11:     if sup > min_sup then
12:       q=buildConfidenceQuery(C[i])
13:       if cHashT.find(q) = NULL then
14:         conf=executeQuery(q)
15:         insert(cHashT, q, conf)
16:       else
17:         conf=cHashT.find(q)
18:       end if
19:       if conf > min_conf then
20:         solC.puhs_back(C[i]
21:       else
22:         sC.push_back(C[i])
23:       end if
24:     end if
25:   else
26:     if sHash.find(q) = NULL then
27:       sup=executeQuery(q)
28:       sHash.insert(q, sup)
29:     else
30:       sup=sHash.find(q)
31:     end if
32:     if sup > min_sup then
33:       q=buildConfidenceQuery(C[i])
34:       if cHash.find(q) = NULL then
35:         conf=calculateSQL(q)
36:         cHash.insert(q, conf)
37:       else
38:         conf=confHash.find(q)
39:       end if
40:       if conf > min_conf then
41:         solC.push_back(C[i])
42:       else
43:         sC.push_back(C[i])
44:       end if
45:     end if
46:   end if
47: end for
48: RETURN sC
```

Algorithm 7 Selective Tabular CRIS: Specialization

Require: C_l: Vector of concept descriptors of length l

Ensure: C_{l+1}:Vector of concept descriptors of length $(l+1)$

Local Variables: specHash: Look-up table

```
1:  for i = 0 to C.size − 1 do
2:    for j = i + 1 to C.size do
3:      if unifiable(C[i], (C[j])) then
4:        uC = unify(C[i], (C[j])
5:        if specHash.find(uC) = NULL then
6:          C_{l+1}.push_back(uC)
7:          specHash.insert(uC)
8:        end if
9:      end if
10:   end for
11:   specHash.clear()
12: end for
13: RETURN C_{l+1}
```

3.4 Applicability of the Approach to other Concept Discovery Techniques

Tabular CRIS, Tabular CRIS-wEF, and Selective Tabular CRIS are proposed to improve running time of ILP-based concept discovery systems without modifying their concept induction mechanism. Each technique improves its predecessor in certain ways while preserving the predecessor's core properties. So each of the proposed approaches can be embodied in an ILP-based concept discovery system in a similar manner. In this section we discuss applicability of Tabular CRIS-wEF to a number of other ILP-based concept discovery systems. We picked four ILP-based concept discovery systems, three of which are top-down systems and one is a bottom-up system, as case studies. Firstly we introduce search space evaluation mechanism of these systems and then propose a method on how to embody the proposed approach.

LINUS [94] is a top-down ILP-based concept discovery system that incorporates first order logic in attribute-value learners such as ID3 [126] and IQ7 family [100]. In ID3 version of LINUS, called ASSISTANT [33], concept descriptors are pruned based on their entropy (3.3).

$$E(\varepsilon) = -\sum_{j=1}^{N} p_j . log_2 p_j \tag{3.3}$$

where ε is the training set, p_j is the prior probability which is computed as

$$p_j = \frac{n^{C_j}}{n} \tag{3.4}$$

n being the size of the training set and n^{C_j} is the number of examples from ε of class C_j.

In NEWGEM [105], which is IQ7 embodied version of LINUS, concept descriptors are pruned based on their coverage. Similar to CRIS, in NEWGEM, coverage refers to the number of positive and negative target instances explained by the concept descriptors. The proposed approach can be incorporated in both applications as follows. For each concept descriptor, its related target instances are stored in a list and as solution clauses are found, target instances explained are removed from the lists. As the lists keep updated results of the concept descriptors, once a concept descriptor is regenerated its result may directly be calculated by counting tuples in its list.

SAHILP [144] is another top-down ILP-based concept discovery system that uses weighted relative accuracy (3.5) as the concept descriptor's quality measure. In relative accuracy measure, the *gain* function is extended with generality and relativity accuracy [125] .

$$WRAcc(H \leftarrow B) = p(B) \times (p(H|B) - p(H)) \tag{3.5}$$

In (3.5) p(B) and P(H) refer to coverage of the body and the head, respectively and p(H|B) refers to the accuracy of the concept descriptor. Coverage measures fraction of instances covered by the body of a rule, and accuracy measures the fraction of predicted positives that are true positives in binary classification problem. In other words, SAHILP employs a quality measure that counts the number of positive and negative target instances. Hence the proposed approach can be integrated in SAHILP in such a way that the modified covering algorithm is employed to update the explained target instances set.

FOIL is one of the earlist ILP-based concept discovery systems and employs a top-down search to construct the concept definitions. As equations (3.6) and (3.7) show, its evaluation functions are about the number of the positive and negative target instances concept descriptors explain. The proposed

approach can be incorporated into FOIL to calculate the goodness value of the concept descriptors by memorizing coverage sets of their parents and updating them as some solution clauses are found.

$$Gain(L_i) = T_i^{++} \times (l(T_i) - l(T_{i+1})) \tag{3.6}$$

$$l(T_i) = -log_2(\frac{T_i^+)}{T_i^+ + T_i^-}) \tag{3.7}$$

GOLEM is a bottom-up ILP-based concept discovery system that builds the bottom clauses by picking up some positive target instances at random and refining them via relative least general generalization operator (rlgg) [110]. Concept descriptors that have the highest coverage are chosen as the solution clauses. Similar to (3.5), in rlgg coverage refers to the target instances explained by a concept descriptor. Therefore, proposed method can be applied on GOLEM in a similar way.

Based on the discussion above we can state that the proposed approaches can be embodied both in top-down and bottom-up ILP systems if they employ quality measures that calculate the number of positive and negative target instances explained by concept descriptors and also employ covering approach.

3.5 Comparison to Other ILP-based Concept Discovery Systems with Memoization

In this section we compare the proposed methods to some state of the art methods, namely Query Packs, Common Prefixes, Coverage List, and Coverage Caching, that employ memioization in ILP-based concept discovery systems for speeding up purposes. In this chapter while addressing

All the above named methods look for partial or exact match among concept descriptors, while with proposed approaches we look for similarities, indeed exact match, among evaluation queries. Looking for common substructures among concept descriptors have certain flaws. As an example consider the two concept descriptors given below
r_1: p(A, B) :- q(C, A), r(D, E)
r_2: p(A, B) :- r(C,D), q(E, A)
Although they are semantically equivalent, methods that look for common substructures within concept descriptors will not be able to detect that these two concept descriptors have the same goodness degree. Tabular CRIS-wEF can handle such situations as they map into the same evaluation query.

Query Packs and Common Prefixes look for similarities with in the concept descriptors that are of the same length. On the other hand two concept descriptors may share the same evaluation query if the larger one differs from the other with literals that have unbounded variables. In such a case Tabular CRIS-wEF can retrieve the goodness degree from the hash table, while Query Packs and Common Prefixes need to execute the entire query.

Tabular CRIS-wEF differs from the Coverage List approach in the way it uses the saved lists. In the Coverage List approach the saved instances are used to calculate the coverage values of refinements of a concept descriptor, while in Tabular CRIS-wEF they are used to calculate coverage values of concept descriptors that map into the same evaluation query.

Common Prefixes, Coverage Lists, and Coverage Caching make use of the underlaying Prolog engine to deal with the look-up tables. Tabular CRIS-wEF is independent of the underlaying data storage and management systems as its memoization mechanism is a part of the concept discovery framework.

CHAPTER 4

PARALLELIZATION TO SCALE UP ILP-BASED CONCEPT DISCOVERY SYSTEMS

Although memoization based methods improved running time of CRIS in magnitudes, the search space evaluation step still constitutes the major part of the total running time of the resultant systems. In Figure 4.1 we plot the execution times of search space evaluation and others for Tabular CRIS-wEF for the Muta, PTE, and PTE-5 data sets.

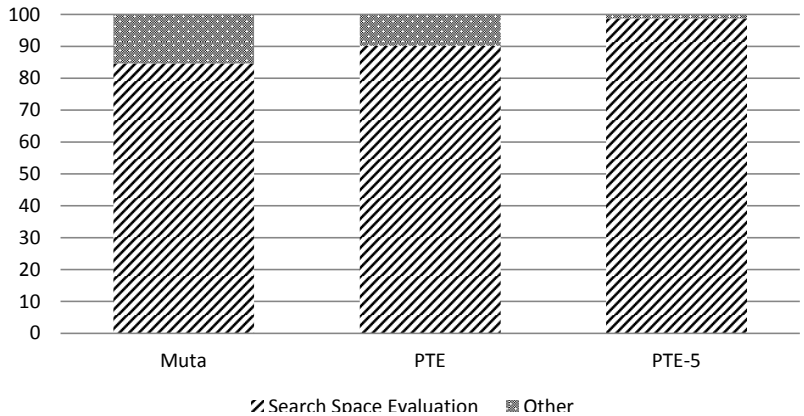


Figure 4.1: Relative running times of components of Tabular CRIS

In this section we further discuss the speeding up ILP-based concept discovery systems and propose methods that incorporate parallelization. Although the proposed methods can be integrated into any ILP-based concept discovery systems in general, we base our discussions to CRIS. Parallelization methods introduced in this section address modifications on the search space formation and the search space evaluation steps of CRIS. The resultant system is named *pCRIS*.

Through this section we will use the*dauter* dataset given in Table 4.1 as a working example.

Assuming that minimum support is set to 0.8, Table 4.2 lists the most general two literal concept descriptors that satisfy the minimum support threshold.

Search space evaluation step of CRIS constitutes majority of the total running time, as it executes many

Table4.1: The daughter dataset

Concept Instances	Background Facts
daughter(mary, ann)	parent(ann, mary)
daughter(eve, tom)	parent(ann, tom)
	parent(tom, eve)
	female(ann)
	female(mary)
	female(eve)

Table4.2: Support and confidence values of two-literal concept rules

R. Index	Concept Descriptor	Support	Confidence
1	daughter(A,B)←parent(B,A)	1	0.666
2	daughter(A,B)←parent(C,A)	1	0.166
3	daughter(A,B)←parent(B,C)	1	0.25
4	daughter(A,B)←parent(C,D)	1	0.125
5	daughter(A,B)←female(A)	1	0.166
6	daughter(A,B)←female(C)	1	0.125

SELECT type of SQL queries. Each such query execution requires as many table joins as the length of the concept descriptor, and in some cases range selections are performed. As no write operations are performed during the search space evaluation step, it can be parallelized by distributing the queries over multiple workers.

Operations of the search space formation are inexpensive, this step may be a bottleneck if there are too many concept descriptors to be specialized though. This step is also well suitable for parallelization as none of the write operations interfere with the read operations.

4.1 Data Dependence Analysis

Data dependence analysis is the study of determining dependence relationships between statements in a sequential or centralized program due to control and data flows [35]. It has originally been proposed for automatic optimization, parallelization and vectorization of sequential programs, later it has been utilized also to be employed in various software development steps such as program maintenance and testing. Such dependencies are broadly classified into three categories:

1. Flow Dependence (δ): S_2 reads a data value which was earlier written by S_1
   $S_1\ \delta\ S_2 \leftrightarrow \exists$ x: x $\in$ OUT(S_1) $\bigwedge$ x $\in$ IN(S_2) $\bigwedge$ $S_1\ \Theta\ S_2$ $\bigwedge \exists!\ (S_1\ \Theta\ S_k\ \Theta\ S_2 \bigwedge$ x $\in$ OUT(S_k))
2. Anti Dependence ($\bar{\delta}$): S_1 reads a data value before it is altered by S_2
   $S_1\ \bar{\delta}\ S_2 \leftrightarrow \exists$ x: x $\in$ IN(S_1) $\bigwedge$ x $\in$ OUT(S_2) $\bigwedge$ $S_1\ \Theta\ S_2$ $\bigwedge \exists!\ (S_1\ \Theta\ S_k\ \Theta\ S_2 \bigwedge$ x $\in$ OUT(S_k))
3. Output Dependence (δ°): S_2 overwrites a data value which is calculated earlier by S_1
   $S_1\ \delta^\circ\ S_2 \leftrightarrow \exists$ x: x $\in$ OUT(S_1) $\bigwedge$ x $\in$ OUT(S_2) $\bigwedge$ $S_1\ \Theta\ S_2$ $\bigwedge \exists!\ (S_1\ \Theta\ S_k\ \Theta\ S_2 \bigwedge$ x $\in$ OUT(S_k))

where

- *IN(S)* is set of all data elements, whose value is read by S
- *OUT(S)* is set of all data elements, whose value is altered by S
- $S_1 \Theta S_2$ states that instance of S_1 can be executed before an instance of S_2

In the rest of this section we analyze the search space formation and search space evaluation steps of CRIS in this respect and propose methods for any occurring dependence.

The Sequential Search Space Formation algorithm, Algorithm 8, is data-independent as all *read* and *write* operations enclosed in the loop are dependent only on the currently executing index.

Algorithm 8 Sequential Search Space Formation

Require: fC:Frequent clauses of length l
Ensure: sC:Possible clauses of length $(l + 1)$

```
1: for i = 0 to fC.size − 1 do
2:    for j = i + 1 to fC.size do
3:       if unifiable(fC[i], fC[j]) then
4:          uC = unify(fC[i], fC[j])
5:          sC.push_back(uC)
6:       end if
7:    end for
8: end for
9: RETURN sC
```

Algorithm 9 Sequential Search Space Evaluation and Pruning

```
1: //posSol is a global variable to store possible solution clauses
```

Require: pC: Possible clauses of length l
Ensure: fC: Frequent clauses of length (l)

```
2: for i = 0 to pC.size do
3:    pC[i] = calculateSupport(pC[i])
4:    if pC[i].sup ≥ min_sup then
5:       fC.push_back(pC[i])
6:    end if
7: end for
8: for i = 0 to fC.size do
9:    fC[i] = calculateConfidence(fC[i])
10:   if fC[i].conf ≥ min_conf then
11:      posSol.push_back(fC[i])
12:   end if
13: end for
14: RETURN fC
```

The Sequential Search Space Evaluation and Pruning algorithm, Algorithm 9, exhibits a flow dependence: $B_5\ \delta\ B_9$ [1] due to *fC*. To deal with this dependence, workers should be synchronized before entering the second loop in the parallel version of the *search space evaluation and pruning* step.

[1] B_n refers to the n^{th} line of Algorithm 9

At functional level, there exists a flow dependence on Algorithm 8 and Algorithm 9. Output of Algorithm 8 is the input of the Algorithm 9. While parallelizing CRIS, workers should synchronize at this point: all workers need to send their partial results back to the master upon finishing the search space formation step and block until receiving a start message for the search space evaluation and pruning step.

4.2 Framework and Design Issues

Parallel version of CRIS employs master-worker architecture. The master - worker paradigm consists of two entities: a master and multiple workers. The master splits a problem into subproblems, dispatches subproblems among workers, and gathers the partial results in order to produce the final result. The workers execute in a very simple cycle: receive a start message from the master, process the task, send the result back to the master, and wait for the next start message [77]. The literature contains several successful parallel applications based on master - worker paradigm [14].

pCRIS is designed to work in a shared-nothing environment. Each processor has its private memory and disk. Communication is based on message passing. Each computational entity needs to access to the entire data set. As distributing the data will be expensive, all data should be made available on processor's disk beforehand.

pCRIS is parametric in such a way that user can set

a) which steps to run in parallel

b) initial chunk sizes in parallel execution

c) minimum data size for parallel execution of a step

Minimum data size for parallel execution of a step is a distinctive property of pCRIS. Based on this parameter, for different calls in a run, a step may work either in sequential mode or parallel. This aims to eliminate situations where communication cost may overwhelm computational cost. Similarly, a step may start running in a parallel mode but later, as the data size decreases, workers may stop executing and master executes the remaining computations. The *search space formation* and the *search space evaluation and pruning* steps of CRIS are parallelized in pCRIS. Figure 4.2 shows the overall flow of pCRIS. Rectangles with dashed borders present the work done by the workers while rectangles with straight borders show the work done by the master. pCRIS retains all properties of Tabular-CRIS.

4.3 Parallelizing the Search Space Construction Step

In the search space formation step, unifiable rules of length l are concatenated in order to form rules of length $l+1$. In order to prevent redundancy, rules are grouped with respect to the rules they are generated from and each group is given an index number. The candidate rules are generated in such a way that a rule is unified with rules that have an index number greater than the index number of the target rule.

Parallel version of the *search space formation* follows the ideas from the Count Distribution Algorithm [13]. Each worker should specialize a mutually exclusive subset of the candidate clauses. To yield a

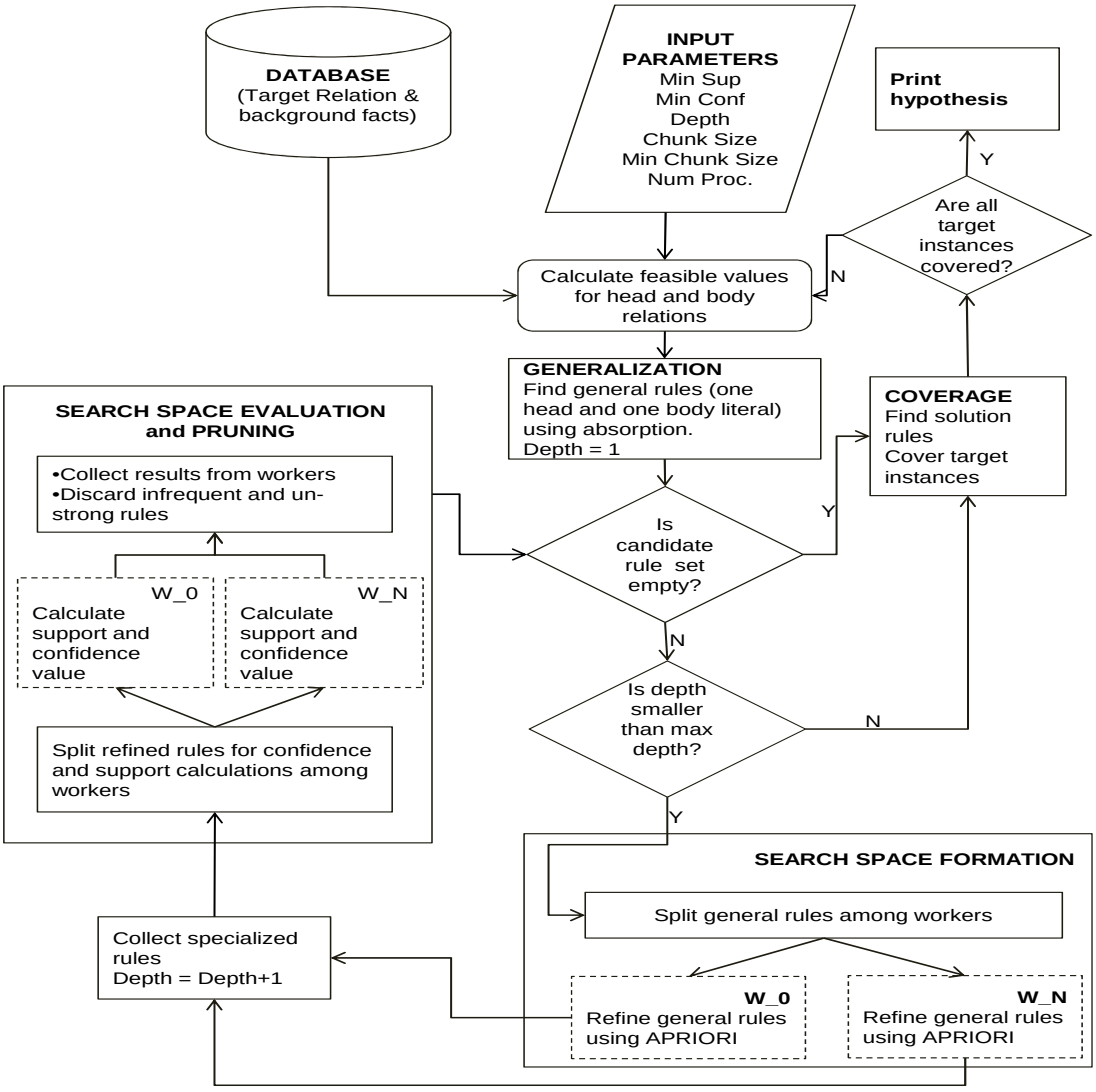


Figure 4.2: The flowchart of parallel system

good work balance, number of the candidate rules should be evenly divisible by the number of workers. Master calculates the index number of the last rule to be specialized in parallel, and sequentially specializes the rules that have larger indices.

At the beginning of the parallel search space formation step, master calculates the last clause index to be specialized in parallel, the *ParallelWorkingEndRange* function. The index is calculated based on the number of clauses to be specialized, *cl*, and the number of workers, *ws*. Master broadcasts the candidate clauses and the last index to be specialized. To specialize a rule at index *i*, each worker calculates its working range based on its rank, *r*, and the number of the clauses, *cl_s*. The functions *calculateStartIndex* and *calculateEndIndex* given in Figure 8 implement these calculations. Workers send the specialized clauses back to the master. Master specializes the remaining candidate clauses and removes the repeating ones. All communication among the processors is based on blocking routines.

For the specialization of the rules given in Table 4.2, workers and the master will specialize the clauses as given in Table 4.3.

Table4.3: Index ranges that each worker takes in *specialization* step

Rule Index	W_1	W_2	W_3	**Master**
1	[2, 3]	[4, 5]	[6]	–
2	[3, 4]	[5]	[6]	–
3	[4]	[5]	[6]	–
4	[5]	[6]	–	–
5	[6]	–	–	–

Given two candidate clauses C_1:$h_1 \leftarrow b_1$ and C_2:$h_2 \leftarrow b_2$ are unifiable if

a) C_1 and C_2 share the same head predicate

b) b_1 and b_2 differ by at most 1 literal

Since CRIS finds rules for a specific target concept, all generated rules have the same head predicate. Two literals are equivalent if they share the same relation name and the same bounded variables at the same argument places.

CRIS applies a substitution to the literal in order to rename variables in the literal according to the variables in 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. Specialization of the candidate clauses given in Table 4.2 results in 24 new possible clauses of length 2. Flowchart of the algorithm for parallelized version of the specialization step is given in Figure 4.2.

4.4 Parallelizing the Search Space Evaluation Step

The search space evaluation and pruning step takes possible clauses as input, maps them into SQL queries and calculates their support and confidence values. According to the support value, the infrequent clauses are pruned. Two different actions can be taken on the basis of the confidence value. If confidence score satisfies the *min_conf* criteria, it is added into the set of possible solutions clauses. Otherwise, it is added to next level's possible clauses vector for further consideration.

As described in [112], different rules may map to the same SQL queries. This is due to the different substitutions of variables of the same type within the same predicate. Tabular CRIS solves this problem by making use of look-up tables. With a single processor, Tabular CRIS works well since the repeating queries will eventually be a hit in the look-up table. In parallelized algorithm, repeating queries may be sent to different processors, resulting in redundant work. To be able to avoid such situations, master maintains a vector of distinct queries, and distinct queries are distributed among workers. By this approach, workers do not require to build their own look-up tables for repeating queries. Once all distinct SQL queries are executed on workers and results are gathered in the master, rules are assigned with their corresponding support and confidence values.

In this step, firstly the support and then the confidence scores of a clause are calculated. This order is followed because a) confidence values calculation requires execution of two queries while support

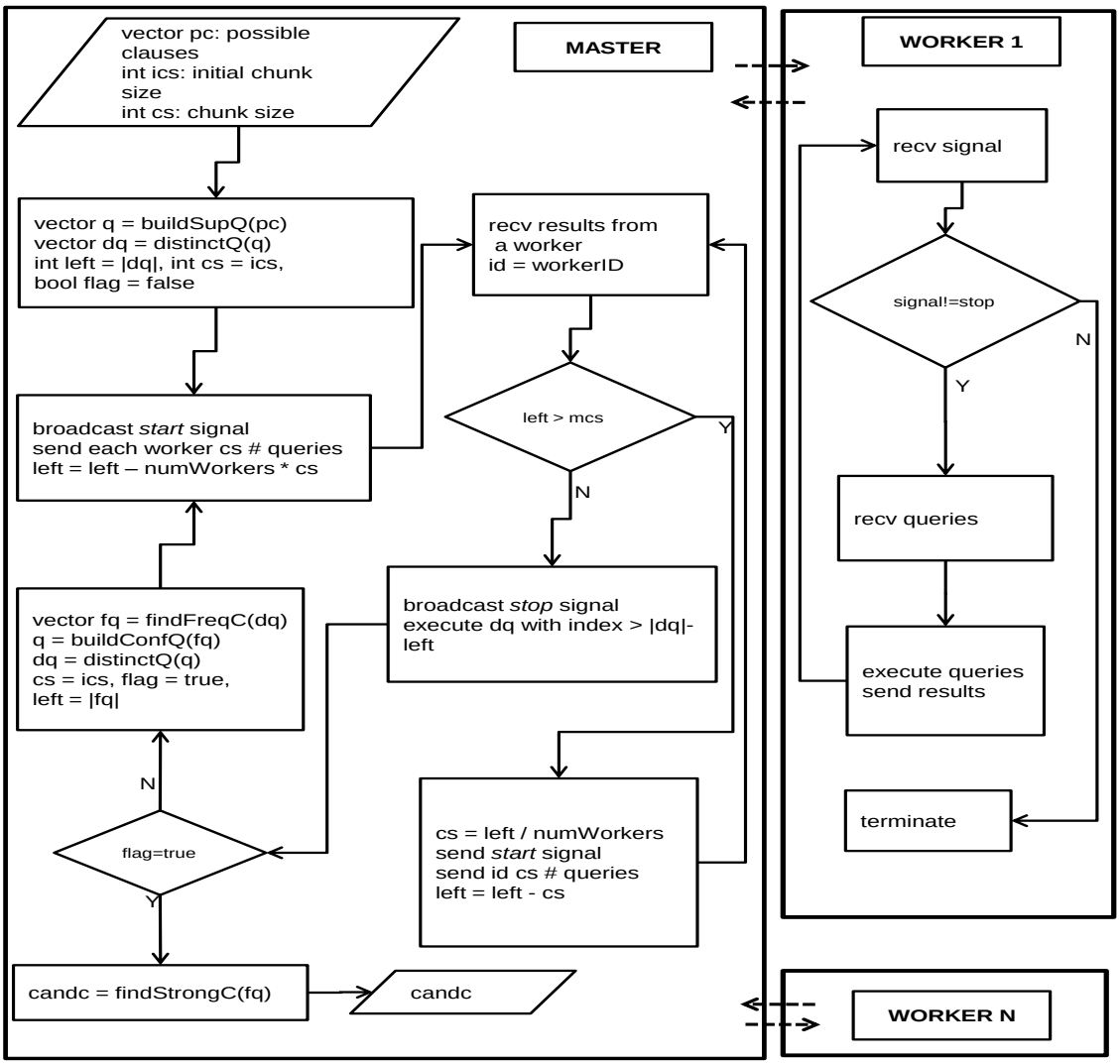


Figure 4.3: The flowchart of parallel search space evaluation and pruning algorithm

value calculation requires execution of a single query, b) a confident clause should be a frequent clause as well.

While parallelizing the search space evaluation and pruning step, workers need to synchronize upon finishing calculation of the support values. Master collects support results from the workers, prunes the infrequent clauses and builds the confidence queries of the frequent clauses. Distinct queries are determined and these queries are distributed among the workers.

Since query execution times vary, distributing queries among workers in equal size of chunks may lead to idle processors. Both for the support value calculation and confidence value calculation, master sends a user specified number of queries to each worker. Any worker that has finished its chunk requests for a new chunk of queries. New chunk sizes are computed at running time and they are monotonically decreasing. Sending chunks in different sizes is important as it decreases the possibility for workers to block on master while sending results back. Another concern is to enable master to stop distributing the queries when the chunk size drops below a threshold as communication cost may

Table4.4: Parallel Search Space Evaluation and Pruning Output for the *Daughter* Example

	Source	**Destination**	**Action**			**Source**	**Destination**	**Action**
1	Number of distinct support queries : 14				16	Worker 2	Master	Send 4 queries
2	Master	Worker 1	Send 4 queries		17	Master	Worker 2	Send 4 queries
3	Master	Worker 2	Send 4 queries		18	Worker 1	Master	Send 4 queries
4	Master	Worker 3	Send 4 queries		19	Master	Worker 1	Send 3 queries
5	Worker 1	Master	Send 4 queries		20	Worker 3	Master	Send 4 queries
6	Worker 2	Master	Send 4 queries		21	Master	Worker 3	Send 2 queries
7	Worker 3	Master	Send 4 queries		22	Worker 2	Master	Send 4 queries
8	Master is executing query at index 12				23	Master	Worker 2	Send 1 query
9	Master is executing query at index 13				24	Worker 1	Master	Send 3 queries
10	Number of distinct confidence queries : 29				25	Master	Worker 1	Send 1 query
11	Master	Worker 1	Send 4 queries		26	Worker 3	Master	Send 2 queries
12	Master	Worker 2	Send 4 queries		27	Worker 2	Master	Send 1 query
13	Master	Worker 3	Send 4 queries		28	Worker 1	Master	Send 1 query
14	Worker 1	Master	Send 4 queries		29	Master is executing query at index 27		
15	Master	Worker 1	Send 4 queries		30	Master is executing query at index 28		

overwhelm computation cost.

Flowchart of the parallel algorithm for search space evaluation and pruning step is given in Figure 4.3. *buildSupQ()* and *buildConfQ()* functions translate the concept descriptors into support and confidence queries, respectively. *dq* is allocator to store the unique queries returned by the *distinctQ()* function. *findFreqC()* prunes the infrequent clauses and *findStrongC()* appends the non-strong clauses into the search lattice for further specializations and the strong clauses into the solution set. As in the parallel search space formation algorithm, the search space evaluation and pruning algorithm implements the communication with blocking routines.

For the *daughter* example, master receives 24 possible clauses as input and translates them into the support queries. 14 of these queries are distinct and 10 are repeating. Output of the parallel search space evaluation and pruning step for the *daughter* example is given in Table 4.4. This output corresponds to execution with four processors, one master and three workers, initial chunk size set to 4 and minimum data size set to 2. Master starts with dispatching query chunks with user specified size, in this case 4, to every worker and waits for the results. As there are 14 queries to be executed and minimum chunks size is 2, master does not send any other support query to workers but instead executes the remaining two queries itself. After this step, it removes the infrequent clauses and builds SQL queries for confidence score calculation. Master sends 4 queries to each worker and goes to listening phase for results. Upon starting to receive results, master sends new chunks of queries in decreasing sizes, in this example chunks are of size 3, 2 and 1. When there are less than minimum chunk size number of queries to be executed, master sends *stop* signal to each worker and executes the remaining queries.

CHAPTER 5

GRAPH-BASED CONCEPT DISCOVERY

In this chapter, we present a hybrid graph based concept discovery system and investigate the effect of the proposed structure for running time improvement. It is a hybrid approach such that it looks for paths, which eventually form the concept descriptors, on a compressed graph. The proposed approach distinguishes from other state of the art graph based approaches by inducing the concept descriptors while building the graph. In this approach, in addition to being the representation framework, the graph structure is also used to guide the concept induction process.

Among the others, a common problem faced by ILP-based concept discovery systems is the so called locale plateau problem [128], where refinement by *one literal at a time* operators of ILP are insufficient to improve a rule quality. As to our knowledge, graph based approaches were first proposed to solve this problem by Richards and Mooney [133], which replaces the refinement by one literal a time operators of ILP with refinement by *multiple literals at a time* operators. Theoretical foundation on the relations among graphs, hypergraphs, and inductive logic programming are provided in [26].

Graph-based concept discovery methods can be classified into two main categories: substructure based approaches and path-finding based approaches. Substructure based approaches [38, 70] rely on the idea that if there exists some concepts in a graph then they should appear as frequent substructures. Such systems repeatedly look for substructures that best compress the graph and replace such substructures with a new vertex. Metrics such as minimum description length [149], information gain [165] are employed to evaluate the candidate substructures.

The motivation behind the path-finding based approaches [66] is the assumption that concepts should appear as frequent, finite length paths that connect some arguments of positive target instances. Such systems usually put constraints on the source and the target vertices and look for frequent paths that connect such pairs of nodes. To evaluate the discovered paths metrics such as F1-measure [163] are employed.

In this work we propose a hybrid framework that combines properties of substructure based and path-finding based approaches of graph-based concept discovery systems. Our work is similar to structure based approaches in a way that instead of representing each fact as a distinct vertex in the graph, it groups the similar facts and represents them as a single vertex in the graph. Similar to path-finding approaches, it infers the concept descriptors by finding paths that connect arguments of target target instances. Different than state of the art graph-based concept discovery systems, the proposed method discovers the concept descriptors while constructing the graph. The current implementation of the proposed approach is limited with working on binary relations that belong to the Head Output Connected (HOC) class of learning problems [142].

5.1 Proposed Method

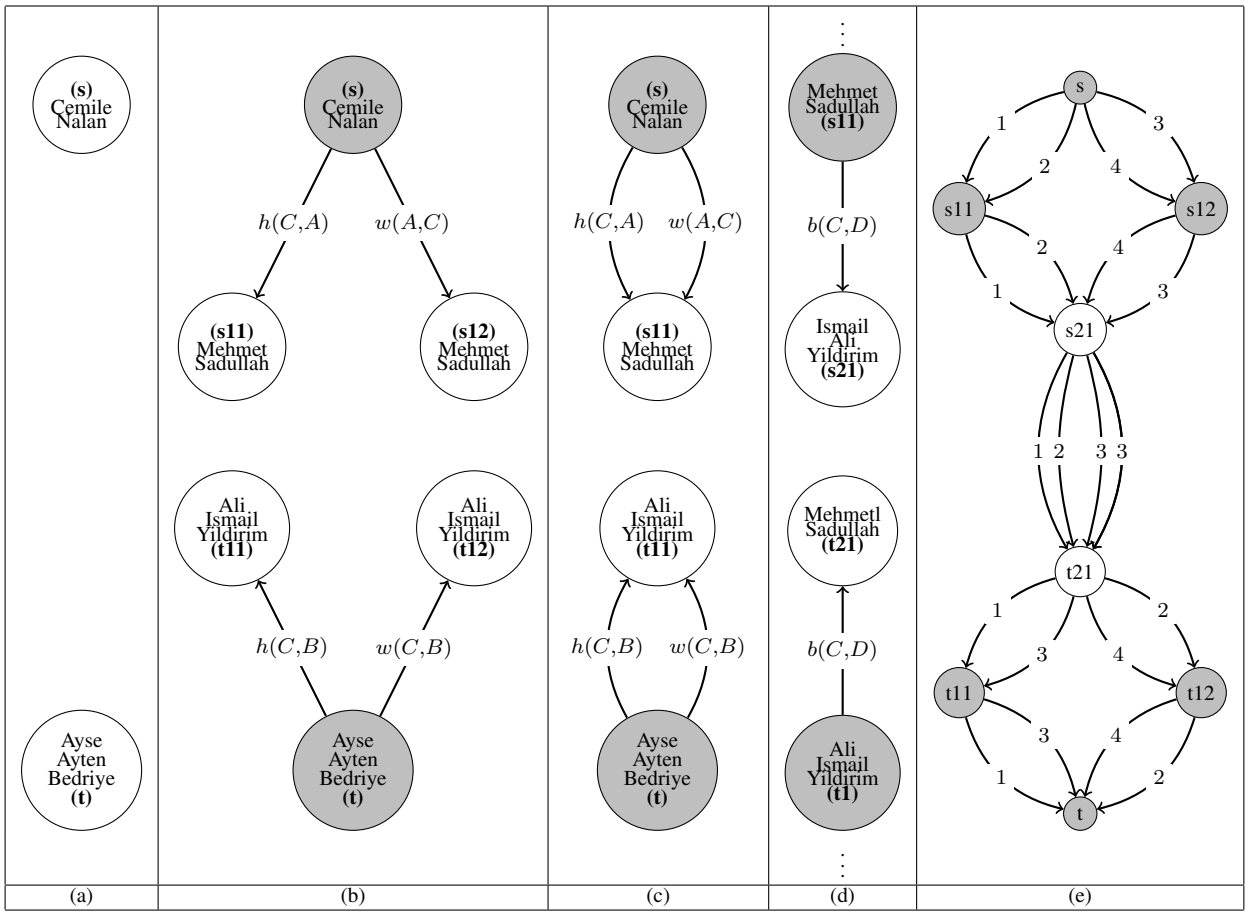


Figure 5.1: Execution of the Algorithm

In this section we firstly discuss our data representation model. Next, we present the concept discovery process, and lastly we discuss the correctness of the proposed approach.

We employ the *elti* data set given in Table 5.1 as a running example in this section. In the data set, predicate *e* stands for *elti* relation, *h* stands for *husband* relation, *w* stands for *wife* relation, and *b* stands for *brother* relation. All arguments are of type *person*.

In Figure 5.1, we illustrate execution steps of the algorithm. Gray vertices are expanded, and white vertices are yet unexpanded. Text given in braces in the vertices are vertex labels and in this example are used for illustrative convenience.

Table5.1: The *elti* data set

Target Instances	**Backgorund data**	
e(cemile, ayse)	b(mehmet, ismail)	h(mehmet, cemile)
e(cemile, ayten)	b(mehmet, ali)	h(ismail, ayten)
e(nalan, bedriye)	b(sadullah,yildirim)	h(ali, ayse)
	h(sadullah, nalan)	w(ayten, ismail)
	h(yildirim, bedriye)	w(ayse, ali)
	w(cemile, mehmet)	w(nalan, sadullah)
	w(bediye, yildirim)	

5.1.1 Data representation

We employ directed acyclic graph as a representation framework. Vertices in the graph hold the constants that are valid for the newly introduced variables. Edges connect vertices such that the tail vertex is directly related to the head vertex through some relation given in the background data. Edges are labeled after the relation that the constants in the tail vertex and the head vertex hold for.

5.1.2 Method

The proposed approach inputs a set of target instances, a set of background data; minimum support, minimum confidence, and maximum rule length parameters. The target instances and the background data are initially stored in a database.

In the proposed approach concept descriptors are evaluated and pruned based on their support and confidence values. Similar to CRIS, support of a concept descriptor is the ratio of the number of target instances captured by the concept descriptor over total number of the target instances. Confidence refers to the number of target instances that correctly hold for the rule divided by the number of instances that hold for the body of the rule. Maximum rule length parameter limits the length of the concept descriptors. Concept descriptors that do not qualify the minimum support value are pruned. Concept descriptors that qualify the minimum support but fail with the minimum confidence are further refined in the next iteration. Concept descriptors that qualify both values are added to the solution set. Any path that exceeds the maximum rule length parameter is pruned.

The proposed method builds the graph in compressed manner by grouping arguments that hold for the same relation. It considers the uncovered target instances all together so that avoids the target instance ordering problem. The grouping of the facts also allows embodying the generalization step of concept learning process into the graph based concept induction process - on the contrary to many path-finding based systems where the generalization step is performed as a post-processing step. The proposed approach utilizes absorption operator of inverse resolution for generalization of concept instances.

The proposed method is composed of seven main components:

1. **Initialization:** In this step the source and the target vertices are created, namely *s* and *t*, respectively. This very initial graph is a disconnected graph with two vertices where vertex *s* stores the values of the first argument of the target instances, and the vertex *t* stores the values of the second argument of the target relations, Figure 5.1-a.

2. **Expansion:** In this step graph is expanded by adding vertices that contain related facts of yet unexpanded vertices. To find such related facts, constants in the tail vertices are expanded with the relations which do not appear as labels on their incoming edges as such expansions will create loops. As an example in Figure 5.1-c we do not extend vertex *s11* with the *husband* and *wife* relations but only with the *brother* relation.

3. **Merge:** In this step vertices that store the same content are represented as single vertex. Edges are rearranged accordingly. The graph in Figure 5.1-b turns to be the one in Figure 5.1-c once this step is executed.

4. **Evaluation and pruning:** In this step support and confidence values of the current concept descriptors are calculated. To calculate these values current concept descriptors are translated into SQL queries and these queries are run against the database. In Equations 5.1 to 5.4 we provide evaluation queries for the concept descriptor
elti(A, B):-husband(C,A), brother(C,D)

```
SELECT COUNT(CONCAT(r1.name2,'-',r3.name)
FROM elti r0, husband r1, brother r2, person r3
WHERE r1.name2=r0.name1
AND r3.name=r0.name2
AND r1.name1=r2.name1
```
(5.1)

```
SELECT COUNT(CONCAT(r1.name2,'-',r3.name)
FROM husband r1, brother r2, person r3
WHERE r1.name1=r2.name1
```
(5.2)

```
SELECT COUNT CONCAT(r0.name1,'-',r0.name2)
FROM husband r1, brother r2, elti r0
WHERE r0.is_covered_aux = 0
AND r1.name1=r2.name1
AND r1.name2=r0.name1
```
(5.3)

```
SELECT COUNT CONCAT(r0.name1,'-',r0.name2)
FROM elti r0
```
(5.4)

$$Confidence_C = \frac{Equation 5.1}{Equation 5.2}$$

$$Support_C = \frac{Equation 5.3}{Equation 5.4}$$

Pruning is performed as described above.

5. **Check for intersection:** To find intersections, tail vertexes and their heads of the subgraph with the root node *s* are compared to tail vertices and their heads of the subgraph with the root node *t*. An intersection in a comparison means that such pairs connect the two subgraphs, hence form a path from one argument of the target relation to the other. Such paths are added into the candidate solution set. In the intersection step we allow partial matches, i. e. two tail nodes need not necessarily contain the same constants. By this way we can discover concept descriptors in a noisy data. Similar mechanisms are also employed in structure based approaches, i. e. matches with partial distortions are allowed [39]. In Figure 5.1-d such an intersection exist and the formed paths are:
p1: e(A,B), w(A,C), b(C,D), b(C,D), h(C,B), e(A,B)
p2: e(A,B), h(C,A), b(C,D), b(C,D), h(C, B), e(A,B)

p3: e(A,B), w(A,C), b(C,D), b(C,D), w(C,B), e(A,B)
p4: e(A,B), h(C,A), b(C,D), b(C,D), w(C,B), e(A,B)

In this step the concept discovery process terminates if the length of the current paths is larger than $\lceil max\_depth/2 \rceil$ and no intersection is found.

6. **Update path variables:** In this step the variables of concept descriptors discovered are updated to be consistent with the constant values they hold for. For this purpose we build a new substitution map based on the constants. As an example in partial path *e(A,B), w(A,C), b(C,D)* *D = {Ismail, Ali, Yildirim}*, while in the same constant set is represented with *C* in the partial path *b(C,D), h(C,B), e(A,B)*. To build the final concept descriptors repeating literals are removed from the paths, and variable names are updated to be consistent with each other. The candidate solution set will be:
path1: e(A,B):-h(C,A), b(C,D), h(D,B)
path2: e(A,B):-h(C,A), b(C,D), w(B,D)
path3: e(A,B):-w(A,C), b(C,D), h(D,B)
path4: e(A,B):-w(A,C), b(C,D), w(B,D)

   In Figure 5.1-e we show the traversals of the solution clauses. Once final candidate solution set is constructed, the candidate solution clauses are pruned based on their support and confidence value. Any candidate solution clause that does not qualify the minimum values is pruned. Similar to CRIS, to choose the best candidate concept descriptor f-metric is employed.

   In this step the concept discovery process terminates if the newly discovered concept descriptors have already been discovered in the previous iterations.

7. **Covering:** In this step target instances explained by the solution clauses are marked as *covered*. If the number of the remaining uncovered target instances is below $min\_sup \times \#target\_instances$ the concept induction process terminates, else restarts with the initialization step.

5.1.3 A Discussion on the Proposed Approach

The proposed approach starts with a data set where all target instances are uncovered. Expansion of the graph is achieved by adding new vertices with constants that are related to previously generated vertices. Only those vertices that do not qualify the minimum support and minimum confidence values with respect to the current number of uncovered target instances are pruned. If a vertex is pruned at some iteration due to insufficient support or confidence values, it will be regenerated at a later iteration and at that step if it qualifies the minimum support and minimum confidence values it will participate in a solution clause.

Concept descriptors are formed by conjunctions of relations that have arguments directly or indirectly related to the target instances. In the formation of the graph we include vertices that are connected to some other already created vertices. In this approach some background data may not be included in the graph but as such background data is not directly or indirectly related to the target instances their inexistence in the graph will not hurt the concept induction process.

Concept discovery process terminates only when no new concept descriptors can be found with the current uncovered target instances or remaining number of the uncovered target instances are below of a threshold, i.e. less than $min\_sup \times \#target\_instances$.

CHAPTER 6

EXPERIMENTS

A set of experiments were conducted to evaluate the performance of the proposed methods on well know data sets. The experimental results are discussed in comparison to CRIS and other state of the art methods.

The section starts with the introduction of the experimental environment, followed by the explanations of the metric employed to evaluate the proposed methods. Then we explain the data sets used in the experiments. In the last subsection we present and discussed the experimental results.

6.1 Experimental Environment

The experiments are conducted on Middle East Technical University Computer Engineering Department's (METU CEng) high performance computing facility NAR [1]. NAR has 46 computational nodes (HP ProLiant BL460c) with total of 368 cores, 736 GB of memory and 6.5 TB disk space.

The proposed methods are implemented in C++. MySQL version 5.0.95 [2] is used as the data store and management tool. Due to the resource limitations, while testing pCRIS, instead of installing a dedicated MySQL server for each worker, we run 8 instances of MySQL. Each instance has its own communication port and a separate data path. For the experiments in which more than 8 processors are employed, multiple connections are established to an instance of MySQL over the same port. In pCRIS, Boost.MPI library version 1.37.0 [3] is used as the message passing interface.

6.2 Evaluation Metrics

- **Speedup:** Speedup is metric to measure how much a version of a program is faster then its another version. In case of parallel programs, speedup, S_p, is calculated by dividing the execution time of the sequential program, T_1 by the execution time of the parallel program on p workers, T_p.

$$S_p = \frac{T_1}{T_p} \tag{6.1}$$

  The speedup is called *linear* or *ideal* if $S_p = p$ and *superlinear* if $S_p > p$. Such speedups are generally unachievable due to the shared resources, communication cost, and not fully parallelizable sequential codes [57].

  The same metric has also been widely employed to calculate the efficiency of ILP-based concept discovery systems that employ memoization. In this context speedup is calculating by dividing

the execution time of the original implementation by the execution time of the memoization based version.

- **Efficiency:** Efficiency is a metric to calculate the utilization of the processors in a parallel architecture.

$$E_p = \frac{S_p}{p} \tag{6.2}$$

Efficiency is called *ideal* if $E_p = \alpha \times n$, for some constant α, $0 < \alpha \leq 1$.

- **Coverage:** Coverage donates the number of target instances of the test data is covered by the induced solution clauses over the number of all target instances.

- **Accuracy:** Accuracy donates the fraction the of the sum of correctly covered true positive and true negative target instances over the total number of true positive, true negative, false positive and false negative target instances.

- **Precision:** Precision is the fraction of correctly covered true positive target instances over of the sum of true positive and true negative target instances.

- **Recall:** Recall is the fraction of the correctly covered true positive target instances over the number of true positive and false negative target instances.

- **Hit Count:** Number of searches that do not result with a miss on a hash table.

As CRIS and the proposed extensions to it work on positive only data, the negative examples are generated under the Closed World Assumption.

Not all of the metrics listed above are employed in the evaluation of each experiment, but are selected based on the evaluation metrics employed in the reference materials.

6.3 Data Sets

To evaluate the the proposed methods we conducted experiments on 16 data sets, each of which addresses different aspects of the concept discovery problem. In Table 6.1 we list the properties of these data sets and the experimental settings.

Table6.1: Data sets and experimental settings

Data Set	**Num. Pred.**	**Num. N. Pred.**	**Num. Ins.**	**Min. Sup.**	**Min. Conf.**
Dunur	9	0	224	0.3	0.7
Eastbound	12	0	196	0.1	0.1
Elti	9	0	224	0.3	0.7
Mesh	26	0	1749	0.1	0.7
Muta	8	0	16544	0.1	0.7
PTE	27	0	23850	0.1	0.7
Muta-S-Aggr	12	3	190	0.3	0.7
Muta-4-Aggr.	14	4	17652	0.3	0.7
PTE-5-Aggr.	32	5	29267	0.1	0.7
Student Loan	10	2	5288	0.1	0.7
m5l13	10	2	5288	0.1	0.7
m6l20	10	2	5288	0.1	0.7
m13l31	10	2	5288	0.1	0.7
m15l29	10	2	5288	0.1	0.7
m6l26	10	2	5288	0.1	0.7
m6l28	10	2	5288	0.1	0.7
Family	10	2	5288	0.1	0.7
Same Generation	10	2	5288	0.1	0.7

Elti [81], *Dunur* [81] and *Same Generation* [81] are real life data sets that contain family relationships. The *Elti* data set contains background data that are directly related to the target relation. The *Dunur* data set contains background data that is indirectly related to the target relation. The *Same Generation* data set contains recursive relations. The *Family* [5] data set is a highly relational data set that also contains family relationships.

The *Eastbound* [99] problem aims to predict if a train is eastbound or westbound train based on the trains' properties such as shape and load.

Mesh [48] is an engineering problem data set that aims to determine the mesh resolution values of edges of physicals structures.

Muta [9] is biochemical data set and aim with this problem is learning if a chemical is mutagenic or not. *Muta-4-Aggr.* is extended version of Muta with aggregate predicates. *Muta-S-Aggr.* is a subset of Muta-4-Aggr. The following aggregate predicates are added to the Muta data sets:

- *atm atm(drug,cnt).*
- *atm bond(drug,cnt).*
- *atm count(drug,cnt).*

- *bond count(drug,cnt).*

PTE [152] is a biochemical data sets and aim with this problem is learning if a chemical is carcinogenic or not. *PTE-5-Aggr.* is extended version of PTE with aggregate predicates. The following predicates are manually calculated and added to the PTE data set:

- *pte atm count(drug,cnt).*
- *pte bond count(drug,cnt).*
- *pte atm bond count(drug,cnt).*
- *pte atm charge max(drug,mx).*
- *pte atm charge min(drug,mn).*

The *Student Loan* [123] data set contains data about students' enrollment and employment status, and the aim is to find rules that define a student's obligation for paying his/her loan back.

m5l13, *m6l20*, *m13l31*, *m15l29*, *m6l26*, *m6l28* are data sets from Botta's Phase Transition data set [29, 28]. Phase Transition [68] divides the problem universe into three subspaces such that problems falling in the first subspace, called *yes* region, have many solutions so it is easy to find one, problems falling in the second subspace, called *no* region, have very few solutions if not none, and lastly probability of having solutions for the problems in the third subspace, called *mushy* region, suddenly changes from 1 to 0. *m5l13* and *m6l20* belong to the *yes* region; *m13l31* and *m15l29* belong to the *no* region; and *m6l26* and *m6l28* belong to the *mushy* region.

In the subsequent sections we present the experimental results. The experimental results are

As it is the case with the evaluation metrics, not every method is tested on each data set, but data sets are chosen in accordance to the reference materials.

6.4 Evaluation on Memoization-based Technique

6.4.1 Tabular CRIS

To evaluate the performance of Tabular CRIS, we performed a set of experiments on hit count, running time, and memory consumption. In Table 6.2 we list the hit counts and the memory consumption of Tabular CRIS. The second column list the number of evaluation query hits over the total number of queries generated, the third column list the number of the repeating concept descriptors over the total number of concept descriptors. The last column list the maximum memory consumed during the experiment to store the repeating queries and the concept descriptors.

In Table 6.3 we compare the running time of Tabular CRIS with of CRIS.

As seen in the results, the number of common queries (the number of hits) vary. However, even the smallest number of hits (716 for the Muta data set) provides an important gain. As seen on the third column of the same table, the number of repeating rules vary with the data sets, as well. However, the

Table6.2: Hash Table Hits

Data Set	Query Hits	Repeating Concept Descriptors	Table Size (≈KB)
Dunur	2002/3777	25/1850	28
Elti	946/3571	75/1832	41
Eastbound	9623/21868	0/7294	2
Mesh	141300/154015	323/57100	75
Muta	716/13500	25/6081	166
PTE	9328/25570	346/11008	184
PTE-5-Aggr.	19684/131719	1843/61292	1173
Same Generation	400/996	9/397	8

Table6.3: Running Times.

Data Set	CRIS	Tabular CRIS
Dunur	00:00:26	00:00:01
Elti	00:00:35	00:00:02
Eastbound	00:11:36	00:00:01
Mesh	02:39:34	00:00:32
Muta	00:03:42	00:00:13
PTE	00:21:25	00:08:35
PTE-5-Aggr.	05:17:00	01:37:16
Same Generation	00:00:12	00:00:01

efficiency gain in finding repeating rules is more in this task, since the search complexity is improved even for small number of hits.

Tabular CRIS addresses the search space formation and the search space space evaluation functions of CRIS to improve its running time. In order to analyze the improvement at functional level, in Table 6.4 we list the running times of the search space formation step and the search space evaluation step. Although in majority of the experiments the search space formation step has a longer running time, there are cases where both steps consume almost the same time, i. e. the PTE experiment, and in some cases the total running time is determined by the search space evaluation step, i. e. the Same Generation experiment. These results show that both steps deserve the proposed optimizations.

Experimental results show that the proposed approach is effective and the execution time drops considerable. Independent of the nature of the data set and application, use of hash table improves the complexity of detecting repeating rules from polynomial time to linear time.

6.4.2 Tabular CRIS-wEF

To evaluate the performance of Tabular CRIS-wEF we conducted a set of experiments and analyzed its performance in terms of speedup, hash table hit count, memory consumption, and the cost introduced by the proposed modifications. The experimental results are presented and discussed in comparison to

Table6.4: Running Times at Component Level (hh:mm:ss.s).

Data Set	CRIS		Tabular CRIS	
	S. S. Formation	S. S. Evaluation	S. S. Formation	S. S. Evaluation
Dunur	23.00	03.00	00.20	02.40
Elti	22.00	03.00	00.20	01.32
Eastbound	8:30.00	3:07.00	00.67	05.90
Mesh	02:19:53	10:33.00	02.21	24.00
Muta	01:31.00	11.00	01.80	11.20
PTE	08:27	8:41.00	01.92	02:41.12
PTE-5-Aggr.	03:29:15	01:44:30	21.00	35:01.00

CRIS, Tabular CRIS and some state of the art approaches.

Table6.5: Speedup Comparison

Data Set	Tabular CRIS-wEF	Tabular CRIS
Dunur	13.13	15.38
Eastbound	435	457
Elti	9.85	16.50
Mesh	366	252
Muta-S-Aggr.	16.90	16.38
Muta	14.27	12.07
PTE	7.24	5.90
PTE-5-Aggr.	9.60	8.80
Same Generation	13.1	12

Table 6.5 compares the speedups achieved by Tabular CRIS-wEF to Tabular CRIS over the original implementation of CRIS. As the results indicate for the Dunur, Eastbound and Elti data sets there is a drop in the achieved speedup while there is an increase for the other data sets. Indeed these results are expected due to the following reasons:

- For the Dunur, Eastbound, and Elti data sets the entire solution set that covers all the target instances is found at the end of the first iteration. Tabular CRIS-wEF aims to improve running time of ILP-based concept discovery systems by caching the repeating queries that are generated at different iterations. For these data sets Tabular CRIS-wEF can not benefit from its added functionality but executes more time consuming functions, such as the covering function, compared to Tabular CRIS and achieves less speedup.

- For the PTE data set the entire solution set that covers some of the target instances is found in the first iteration, and a second iteration is executed in order to check whether further solution clauses exist for the uncovered target instances. The second iteration fails to find solution clauses for the uncovered target instances. For the other data sets, the entire solution set is found in multiple iterations. For these data sets Tabular CRIS-wEF benefits from its added functionality and achieves greater speedups compared to Tabular CRIS.

Table6.6: Hash Table Hit Counts

Data Set	Num. Queries	Tabular CRIS-wEF	Tabular CRIS	Improvement
Dunur	3777	2010	2010	-
Eastbound	21868	9623	9623	-
Elti	3571	943	943	-
Mesh	154015	148695	141371	5%
Muta-S-Aggr.	13056	1162	718	38%
Muta	88821	74586	43816	58%
PTE	25570	13282	9347	29%
PTE-5-Aggr.	131719	34194	25132	26%
Same Generation	996	513	400	28%

In memoization based studies, the achieved speedup is directly related to the hash table hit counts. In Table 6.6 we report the number of hash table hit counts. *Num. Queries* column lists the number of queries generated, including the repeating ones. The third and the fourth columns list the number of hash table hit counts due to Tabular CRIS-wEF and Tabular CRIS, respectively. The last column shows the increase in the hash table hit count. As the experimental results show Tabular CRIS-wEF preserves the hash table hit counts for the experiments for which the entire solution clause is found in a single iteration, and has greater hash table hit counts for the data sets for which the entire solution clause is found in multiple iterations.

The greatest increase in hit count is achieved for the Muta data set, 58%, and the smallest increase is achieved for the Mesh data set, 5%. The other data sets have close increases in the hash table hit counts, the average is 30%. To understand why the Muta and the Mesh data sets differ from the other data sets, we examined the structure of the data sets, distribution of the target instances over the class labels, and the solution clauses induced.

- In the PTE and PTE-5-Aggr. data sets there are two classes namely *carcinogenic* and *non-carcinogenic* with 162 and 137 target instances, respectively. Entire set of the solution clauses is found in 2 iterations for both data sets.
- In the Muta data set there are two classes namely *mutagenic* and *non-mutagenic* with 125 and 63 items, respectively. The solution clauses are found in 4 iterations.
- In the Muta-S-Aggr. data set there are two classes namely *mutagenic* and *non-mutagenic* with 9 and 8 target instances, respectively. The solution clauses are found in 2 iterations.
- In the Mesh data set there are 11 class labels with 1, 4, 6, 8, 9, 15, 17, 29, 52, 64 target instances. The entire solution set is found in 10 iterations.

For the Mesh data set Tabular CRIS-wEF either generates concept descriptors with very general heads, i.e. with variable arguments only, or very different ones, i.e. with different constants. Tabular CRIS itself catches the concept descriptors with very general heads, while Tabular CRIS-wEF also catches the ones with constant variables. Concept descriptors that share the same constant at their heads are generated at iterations (1, 4) and (3, 7). The limited increase is due to these hash table hits.

For the Muta data set, the entire solution set is found in 4 iterations. In the first and the third iterations solution clauses with head literal *muta(a,True)*, at the second iteration solution clauses with the head

literal *muta(a,False)*, and at the fourth iteration solution clauses with the head literal *muta(a,b)* are found. Hence the result of the evaluation queries for concept descriptors that are generated at iteration 3 are retrieved from the hash table since they are already generated and evaluated at iteration 1. Similarly results of the evaluation queries for the concept descriptors of iteration 4 are retrieved from the hash table as they are generated in the previous iterations.

Another interesting result is about the Mesh data set again. Although it has a very limited increase in the hash table hit count compared to the other data sets, it demonstrates the greatest speedup. Tabular CRIS is able to catch 91% of the repeating queries and improves the running time of CRIS for the Mesh data set from some 2 hours 39 minutes to 38 seconds. Tabular CRIS-wEF catches 96% of the repeating queries and completes the experiment in around 26 seconds. Although improvement on the running time is around 12 seconds, this results in a great speedup achievement due to the very long execution time of CRIS.

When compared to Tabular CRIS, Tabular CRIS-wEF has two extra costs due to its added functionality. The first cost is the larger memory requirement, and the second one is the extra computational time required to search and remove target instances from the hash table.

In Table 6.7 we report the memory used by Tabular CRIS-wEF and Tabular CRIS, in KB, for the memoization purposes. Tabular CRIS-wEF requires obviously more memory for hash tables. This is due to the fact that Tabular CRIS stores only one integer value per query, while Tabular CRIS-wEF stores a number of strings for each query. The Eastbound differs from the other data sets by requiring very limited extra memory. When we analyzed the evaluation queries for the Eastbound data set, we found out that they consist of at most 3 tuples, while for example the PTE data set each query returns a result set with 56 tuples on the average.

Table6.7: Memory Consumption in KB

Data Set	**Tabular CRIS-wEF**	**Tabular CRIS**
Dunur	46.5	28
Eastbound	2.1	2
Elti	61	41
Mesh	349	75
Muta Small	312	166
Muta	1275	60
PTE	1916	184
PTE-5-Aggr.	12427	1163
Same Generation	143	8

In order to analyze the time required for the extra computations we conducted a number of experiments to measure the time spent on the Covering function. In Tabular CRIS this function marks the explained target instances as *covered* and cleans the hash table for support queries. In Tabular CRIS-wEF, instead of cleaning the hash table, tuples explained by the solution clauses are removed from it. So the modified version of the Covering function has extra cost to search for the explained items in the hash table. In Table 6.8 we compare the time spent for the Covering function. As the results show, for data sets that have relatively long execution times, the time required for the covering function is almost negligible. On the other hand for the experiments that finish in seconds, the extra time is apparent and causes a considerable drop in speedup.

Table6.8: Running time of the Covering Algorithm, time format ss.s

Data Set	**Tabular CRIS-wEF**	**Tabular CRIS**
Dunur	0.837692	0.619531
Eastbound	0.221229	0.139542
Elti	0.038619	0.032761
Mesh	2.578483	2.187224
Muta Aggr.	0.593751	0.365285
Muta	8.602057	7.067541
PTE	2.868113	1.926015
PTE-5-Aggr.	8.063303	5.283680
Same Generation	0.436249	0.238712

Based on the discussions above, we can conclude that Tabular CRIS-wEF considerably improves the efficiency of CRIS, achieving speedups varying from 7.24 to 435. Compared to Tabular CRIS, Tabular CRIS-wEF requires more memory, which is still affordable though, and achieves higher speedups for the problems whose entire solution set is found in multiple iterations. Although it achieves limited speedup over Tabular CRIS, we should note that the main focus of this work is improving hash hit count of Tabular CRIS. Tabular CRIS-wEF improves the hash hit count of Tabular CRIS by around 30%, providing good performance especially for the domains with limited number of concept classes.

In order to further demonstrate efficiency of Tabular CRIS-wEF, we compared its speedup and memory consumption with the state of the art ILP-based concept discovery systems that incorporate memoization. PTE and Muta are the two of the most commonly addressed data sets in ILP-based concept discovery evaluation systems [150, 83, 106, 87, 89], hence our comparison is based on these two data sets.

Table6.9: Comparison of memory consumption of T. CRIS-wEF to other Tabled ILP-based systems

Method Name	**PTE**	**Muta**
T. CRIS-wEF	1.8 MB	0.9MB
Common Prefixes Tabling	11 MB	6 MB
Conjunction of Common Prefixes	Mem. O.F.	6 MB
Query Packs	882 MB	1.5 MB
Query Transformations	2 MB	25 MB
Coverage Caching	12.9 MB	-

Table 6.9 compares memory consumption of such systems. In the table, *Mem. O.F.* means that a memory overflow has occurred. As the experimental results show Tabular CRIS-wEF requires very limited extra memory compared to other systems.

In Table 6.10 we compare Tabular CRIS-wEF to other ILP-based concept discovery systems that employ memoization in terms of the speedup they achieved. The Common Prefixes and Conjunction of Common Prefixes approaches are implemented on APRIL [61] and results are reported in [137]. The Query Packs approach is implemented on WARMR and the results are reported in [21]. The Query Transformations approach is embodied in ALEPH and the speedup results are reported in [41]. The Reordering Literals approach is embodied in ALEPH and its results are reported in [154]. Coverage

caching is implemented in APRIL and its results are reported in [4]. As these experiments are not conducted in the same environment with Tabular CRIS-wEF, we list their results to give an idea about the speedup performance of Tabular CRIS-wEF. As the experimental results show, Tabular CRIS-wEF performs well in terms of achieved speedup and memory consumption.

Table6.10: Comparison of speedup of Tabular CRIS-wEF to other Tabled ILP-based systems

Method Name	**PTE**	**Muta**
T. CRIS-wEF	7.24	14.27
Query Packs	2.51	-
Query Transformations	2.5	4
Reordering Literals	-	1.6

Memoization is a tool to improve efficiency of computer programs without modifying their internals. In case of ILP-based concept discovery systems, incorporating memoization improves their efficiency without effecting the accuracy. In Table 6.11 we compare accuracy of CRIS to ILP-based concept discovery systems that are also referred to in the speedup and the memory consumption comparison experiments. For more detailed comparison of CRIS to the other state-of-the are systems, reader may refer to [82].

Table6.11: Comparison of accuracy

Method Name	**PTE**	**Muta**
CRIS	0.86	0.95
TILDE	0.79	0.85
APRIL	-	0.64

Another way to analyze the accuracy of the relational learners is to test their behavior within the Phase Transition framework [29]. Phase Transition divides the problem universe into three subspaces such that problems falling in the first subspace, called *yes region*, have many solutions so it is easy to find one, problems falling in the second subspace, called *no region*, have very few solutions if not none, and lastly probability of having solutions for the problems in the third subspace, called *mushy region*, suddenly changes from 1 to 0.

In order to analyze behavior of Tabular CRIS-wEF within the Phase Transition framework we conducted a set of experiments on a subset of artificial data sets provided by Botta et. al. [28]. We took equal number of sample data sets from each region. The properties of the induced solution clauses and their accuracy results are compared to that of FOIL reported in [68]. Accuracy of CRIS on the data sets is similar to that of FOIL, slightly higher for the data sets in the *no region*.

For the hash table hit count and the speedup performance, the results for the phase transition data sets are presented in Table 6.12. The second column shows the region the data set belongs, and the third and fourth columns show the total number of queries generated and the number of hash table hit counts, respectively. The last column lists the speedup of Tabular CRIS-wEF over CRIS. The greatest hash table hit count is achieved for the m5l13 data set, for which the entire solution set is found in two iterations. For the all other data sets the achieved hash table hit count is close to each other, around 11%, and the entire solution set for is found in a single iteration. As the experimental results show Tabular CRIS-wEF performs good if the entire solution clause set is found in multiple iterations.

Table6.12: Phase Transition Experiments

Data Set	Region	Num. Queries	Hit Count	Speedup
m5l13	yes	819	380	10.6
m6l20	yes	729	81	3.8
m13l31	no	3081	342	3.6
m15l29	no	4860	342	8.5
m6l26	mushy	312	34	1.5
m6l28	mushy	498	53	2.6

6.4.3 Selective Tabular CRIS

Similar to the evaluation of Tabular CRIS-wEF, we analyzed the performance of Selective Tabular CRIS in terms of memory consumption, hash table hit count, and speedup.

Main purpose of Selective Tabular CRIS is improving the memory consumption of Tabular CRIS-wEF. In Table 6.13 we compare the memory consumption, in KB, of Selective Tabular CRIS to the of CRIS, Tabular CRIS, and Tabular CRIS-wEF for memoization purposes. Columns 5 and 6 compare the change in memory consumption of Selective Tabular CRIS and Tabular CRIS-wEF over Tabular CRIS. A negative percentage indicates a drop in memory usage, while a positive percentage indicates an increase in the memory consumption. The experimental results show that, Selective Tabular CRIS consumes less memory compared to Tabular CRIS-wEF. The drop in the memory consumption is of real significance especially for the data sets that have numerical attributes. For the other data sets drop in the memory consumption is relatively less, i. e. 86% for the PTE-5-Aggr. data set compared to 4.1% for the Mesh data set. For the data sets that have only nominal arguments Selective Tabular CRIS employs Policy 1, Policy 2, and Policy 4. In such data sets saving in the memory is due to the Policy 4, i.e. saving in memory consumption occurs is due to optimization in the specialization step. For the data sets that have numerical arguments Selective Tabular CRIS employs Policy 3 and Policy 4. For such data sets, saving in the memory consumption is due to both steps.

Table6.13: Memory Consumption in KB

Data Set	S-T. CRIS	T. CRIS wEF	T. CRIS	Increase in % w.r.t	
				S-T. CRIS	T. CRIST-wEF
Dunur	41	46.5	28	46%	66%
Eastbound	1.9	2.1	2	-5%	5%
Elti	55	61	41	34%	48%
Mesh	338	349	75	350%	365%
Muta	1264	1273	60	2006%	2021%
PTE	2545	2565	184	1283%	1294%
Muta-S-Aggr	115	312	166	-30%	88%
PTE-5-Aggr.	2727	12427	1163	134%	968%
Student Loan	694	4561	493	40%	825%

In order to analyze how Selective Tabular CRIS behaves with varying number of numerical attributes we conducted a set of experiments on the PTE data set. Staring with 0 aggregate predicates, at each

experiment we added an aggregate predicate to PTE data set and plotted its memory consumption graph.

In order to better see this, in Table 6.14 we list the change in the memory consumption with respect to the memory used with the initial data set. The results show the increase in the memory consumption for S-Tabular CRIS is very small compared to Tabular CRIS. Memory consumption of Tabular CRIS wEF increases by around 384% with the introduction of the fifth numerical argument while the increase of S-Tabular CRIS is only around 7%. The experiments show that S-Tabular CRIS scales well with increasing number of predicates that have numerical attributes.

Figure 6.1: Memory Consumption for Varying Number of Numerical Attributes of the PTE data set

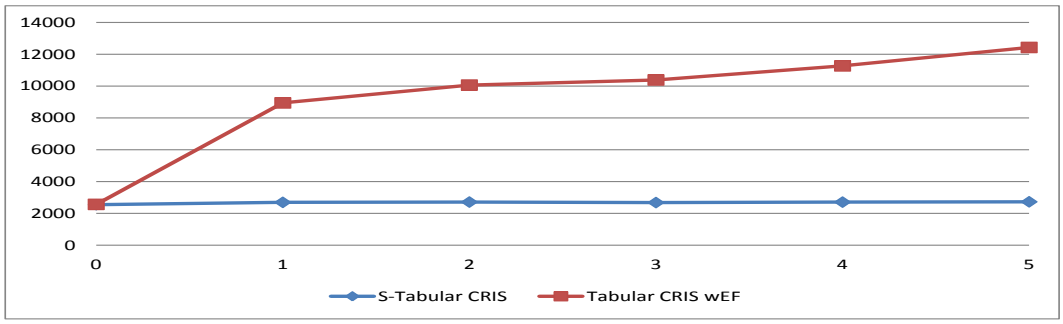


Table6.14: Change in memory consumption

Epoch	1	2	3	4	5
S-T. CRIS	5.8%	6.5%	5.2%	6.4%	7.1%
T. CRIS wEF	248%	292%	304%	339%	384%

Although memory saving achieved by the introduction of the Policy 4 in the specialization step seems relatively small compared to the memory saving in the search space evaluation and pruning step, memory usage in the specialization step may be a problem if there are too many concept descriptors to be refined. One such example is the Muta-4-Aggr. data set. Neither Tabular CRIS nor Tabular CRIS-wEF was able to finalize this experiment. Both Tabular CRIS and Tabular CRIS-wEF encountered memory overflow during the specialization of 10928 concept descriptors. Selective Tabular CRIS was able to pass this step and proceeded with the evaluation of the resulting 775572 concept descriptors. We terminated the execution of S-Tabular CRIS at the point where there were 5 uncovered target instances and 3571087 concept descriptors were generated to explain them. We believe that the induced concept descriptors would be overly specific. Until this point 2787227 concept descriptors were evaluated and only 29677 hash table hits were counted in the search space evaluation and pruning step. This very low number of hash table hit count and being able to proceed two more steps than Tabular CRIS and Tabular CRIS-wEF shows the importance of the optimization introduced by Policy 4 in the specialization step.

In Table 6.15 we list the hash table hit counts of Tabular CRIS, Tabular CRIS-wEF, and Selective Tabular CRIS. The last two columns show the change in the hash table hit counts. The results show that Selective Tabular CRIS does not miss any hash table hit for the data sets that have only nominal

values. For the data sets that have numerical arguments there is a drop in the hash table hit count.

Table6.15: Hash Table Hit Count

Data Set	**S-T. CRIS**	**Tabular wEF**	**T. CRIS**	**Increase in % w.r.t**	
				T. CRIS wEF	**T. CRIS**
Dunur	2010	2010	2010	0%	0%
Eastbound	9623	9623	9623	0%	0%
Elti	943	943	943	0%	0%
Mesh	148695	148695	141371	0%	5%
Muta	74586	74586	43816	0%	70%
PTE	13282	13282	9347	0%	29%
Muta-S-Aggr	946	1162	718	-18%	38%
PTE-5-Aggr.	33202	34194	25132	-2%	25%
Student Loan	791111	834111	811445	-5%	2.7%

In order to understand the drop in the hash table hit count, we analyzed the values the numeric arguments have at each iteration. Table 6.16 lists these values. As in some other studies [43, 140], Selective Tabular CRIS splits the numerical arguments into slots and picks a representative value for each slot. The assumption behind the Policy 3 is that, once the width of a slot changes, its representative value changes as well. Although this applies to many predicates with numerical arguments in the experiments, exceptions are *atom_bond_count* of the PTE-5-Aggr. data set, *atom_count* and *bond_count* of the Muta-S-Aggr. data set, and *absence* and *enrolled* of the Student Loan data set. This is due to the dense repetition of the same numerical value within a relation. For example, in the atom_bond_count predicate there are 340 distinct atoms, each having one the 70 different bond counts. 18 atoms that have 24 bonds, 14 atoms have 18 bonds, 12 atoms have 19 bonds, and the remaining bond counts repeat on average 4 times.

At the end of the each epoch, Selective Tabular CRIS removes evaluation queries of the concept descriptors that have predicates with constant numerical attributes. Because in the above predicates the same constant is densely repeated, a concept descriptor with a previously computed constant value is regenerated, and the query is re-executed. This explains the drop in the hash table hit counts.

In Table 6.17 we list the speedup factors achieved by three versions of CRIS. Speedups are calculated over the running time of the basic implementation of CRIS. As the experimental results show, Tabular CRIS achieves the greatest speedup usually, and the speedup factor drops as the system gets more complicated. Although the drop in speedup for some data sets seems great, this is indeed due to the very much improvement in the execution time. For example for the Mesh data set, CRIS terminates in 2 hours 37 seconds while Tabular CRIS-wEF terminates in 29 seconds and Selective Tabular CRIS terminates in 38 seconds. A very small increase in the execution time, compared to the execution time of the original implementation, results in a great drop of the speedup factor.

Execution time of these system is closely dependent on the evaluation time of the queries. Based on the load of the database server, the evaluation time of the queries may vary. While comparing the speedup factors, we run each test 10 times and took their median as the execution time.

Table6.16: Values of the numerical constant arguments

Data Set	Relation Name	Epoch	Values	Data Set	Relation Name	Epoch	Values
PTE 5 Aggr.	charge	1 2 3	0.047 -0.020 -0.086	Student Loan	Absence	1 2 3 4 5 6 7 8	4 2,4 2,4,6 1,3,5,6 1,3,4,6,7 1,2,4,6,7 1,2,3,4,5,6,8 1,2,3,4,5,6,7,8
	atom count	1 2 3	22 19 18				
	bond count	1 2 3	24 19 18				
	atom bond count	1 2 3	2 2 2		Enrolled	1 2 3 4 5 6 7 8	6 3,6 3,5,8 3,4,7,10 2,4,6,8,10 2,3,4,6,8,10 2,3,4,6,7,9,11 2,3,4,5,6,8,9,11
	max charge	1 2 3	0.540 0.423 0.400				
	min charge	1 2 3	-0.612 -0.639 -0.645	Muta S Aggr.	charge	1 2	-0.117 -0.123,-0.097
Muta-S Aggr.	bond count	1 2	4 3,4		atom count	1 2	4 3,4

Table6.17: Speedup comparison

Data Set	Selective TabularCRIS	Tabular CRIS-wEF	Tabular CRIS
Dunur	12.3	13.3	15.3
Eastbound	368	435	457
Elti	12.3	13.3	16.5
Mesh	190	247	252
Muta	11.93	14.12	12.07
PTE	7.1	7.24	5.9
Muta-S-Aggr.	13.6	13.01	12.59
PTE-5-Aggr.	9.4	9.8	8.8
Student Loan	29.13	40.1	9.17

We also analyzed the statistical significance of the observed speedup factors. For this aim we employed Wilcoxon Mann Whitney test [79]. Wilcoxon Mann Whitney is a non-parametric statistical hypothesis test to determine whether two sets of independent observations have equally large values. Although Wilcoxon Mann Whitney test does not assume any distribution for observations, it assumes that they differ by a location shift (6.3). One possible way to check whether the observations satisfy the shift change condition is to employ Kolmogorov Smirnov test [37].

$$\triangle = \mu(X) - \mu(Y) \tag{6.3}$$

where μ is the median, X and Y are the observation sets, and $\triangle$ is the location shift.

Given the risk factor, i.e. α, the Kolmogorov Smirnov's test computes a *p-value*. *p-value* $\geq \alpha$ indicates that samples satisfy the Wilcoxon Mann Whitney requirement. To justify that a claim is statistically significant, 0.05 is the most commonly used value for α [52].

We run the speed up test tool provided by Touati et al. [157] to statistically analyze the speedups. Table 6.18 presents the result of the analysis. *p-values* of the data sets for the Kolmogorov Smirnov's test are listed in the second column of the table. With the risk factor $\alpha = 0.05$, each data set qualifies to be analyzed by the Wilcoxon Mann Whitney test. Results show that the achieved median speedup is statistically significant.

Median Speedup is the speedup for the median running times: median speedup = $\frac{\overline{med}(X)}{\overline{med}(Y)}$.

Table6.18: Statistical analysis of S-Tabular CRIS' speedup

Data Set	**p Value**	**Speedup Median**	**Is Median Sign.**	**Median Conf Level**
Dunur	0.818	1.002	TRUE	0.5
Eastbound	1	2.671	TRUE	0.99
Elti	1	1.084	TRUE	0.99
Mesh	1	1.299	TRUE	0.99
Muta	0.88	1.917	TRUE	0.94
PTE	0.82	1.68	TRUE	0.99
Muta-S-Aggr.	0.81	1.013	TRUE	0.99
PTE-5-Aggr.	0.82	1.352	TRUE	0.99
Student Loan	0.81	1.607	TRUE	0.99

In Table 6.19 we list the memory consumption of some of the state of the art ILP-based concept discovery systems that incorporate memoization. Experimental results show that Selective Tabular CRIS requires less memory than other systems both for the PTE and Muta data sets.

Table6.19: Comparison of memory consumption of S-Tabular CRIS to other Tabled ILP-based systems

Method Name	**PTE**	**Muta**
Selective Tabular CRIS	2.4 MB	1.2 MB
Common Prefixes Tabling	11 MB	6 MB
Conjunction of Common Prefixes	Mem. O.F.	6 MB
Query Packs	882 MB	1.5 MB

In Table 6.20 we compare the speedups achieved by Selective Tabular CRIS and the state of the art ILP-based concept discovery systems that incorporate memoization. The results show that Selective Tabular CRIS achieves greater speedups than the state of the art systems.

Table6.20: Comparison of speedup of Selective Tabular CRIS to other Tabled ILP-based systems

Method Name	**PTE**	**Muta**
Selective Tabular CRIS	7.4	11.93
Query Packs	2.51	Not Available
Query Transformations	2.5	4
Reordering Literals	Not Available	1.6

As Selective Tabular CRIS does not modify the rule induction of CRIS, its accuracy performance is the same of Tabular CRIS-wEF.

6.5 Evaluation on Parallelization Technique

In this section we demonstrate the experimental results of pCRIS. Due to the resource limitations, instead of running a dedicated worker for each worker, 8 instances of MySQL each with separate communication ports and data paths were run, and in experiments where more than 8 workers were employed more than one connection to a MySQL instance is established. Before discussing the experimental results, we firstly analyze how MySQL instances behaves when more than one connected clients. In Table 6.21 we list its running time of pCRIS on PTE-5-Aggr. with varying number of workers connected to the same MySQL instance. In this experiment chunk and minimum chunks sizes are 180 and 500, respectively.

Table6.21: Running time for connections to a single DB

# Connections	**Running Time**	**Speedup**
1 (Sequential code)	36.13	-
2	30.41	1.18
4	26.20	1.37
9	31.50	1.13

As seen in the results, execution time does not scale well as the number of connections increase. For this reason, we prefer to use a moderate number of multiple connections to a single database instance.

Table6.22: The Best Running Times of the Three Systems

Data Set	**Running Time (hh:mm:ss.s)**			**# Dist. Que. Executed**	**# Processors**	**Chunk Size**
	pCRIS	**Tabular CRIS**	**CRIS**			
Dunur	00:00.90	00:02.88	00:26.00	1775	14	20
Elti	00:00.99	00:02.12	00:35.00	2625	16	30
Eastbound	00:01.88	00:06.54	00:11:36	12245	16	20
Mesh	00:17.12	00:37.36	02:39:34	12715	16	100
Muta	00:04.30	00:12.05	03:42.00	12784	16	50
PTE No Aggr	00:46.00	03:35.00	21:25.00	16242	12	100
PTE 5 Aggr	04:26.00	36:13.00	05:17:00	112035	10	50

In Table 6.22 we present the best running times of pCRIS, Tabular CRIS and CRIS. Although improvement in running time was expected with experiments where large number of queries are executed, speedup is also achieved for the experiments which have very short running times as well.

In order to analyze how pCRIS improves the running times of the parallelized components we conducted a set of experiments. In Table 6.23 we present these results. As the experimental results show pCRIS behaves poorly in the search space formation step. Speedup is not observable in this step as the computations themselves are very cheap and communication cost overwhelms the computation cost. We believe that parallelization of this is step is still necessary especially for experiments which involve high number of concept descriptors to be specialized both for the sake of memory usage and computational efficiency.

Experimental results show that pCRIS shows its power in the search space evaluation step. High speedups are achieved even for experiments, such as Elti and Dunur, where there were relatively less number of concept descriptors to be evaluated, queries distributed as small chunks among many workers.

Table6.23: Running Times of Parallel Components vs Sequential Versions

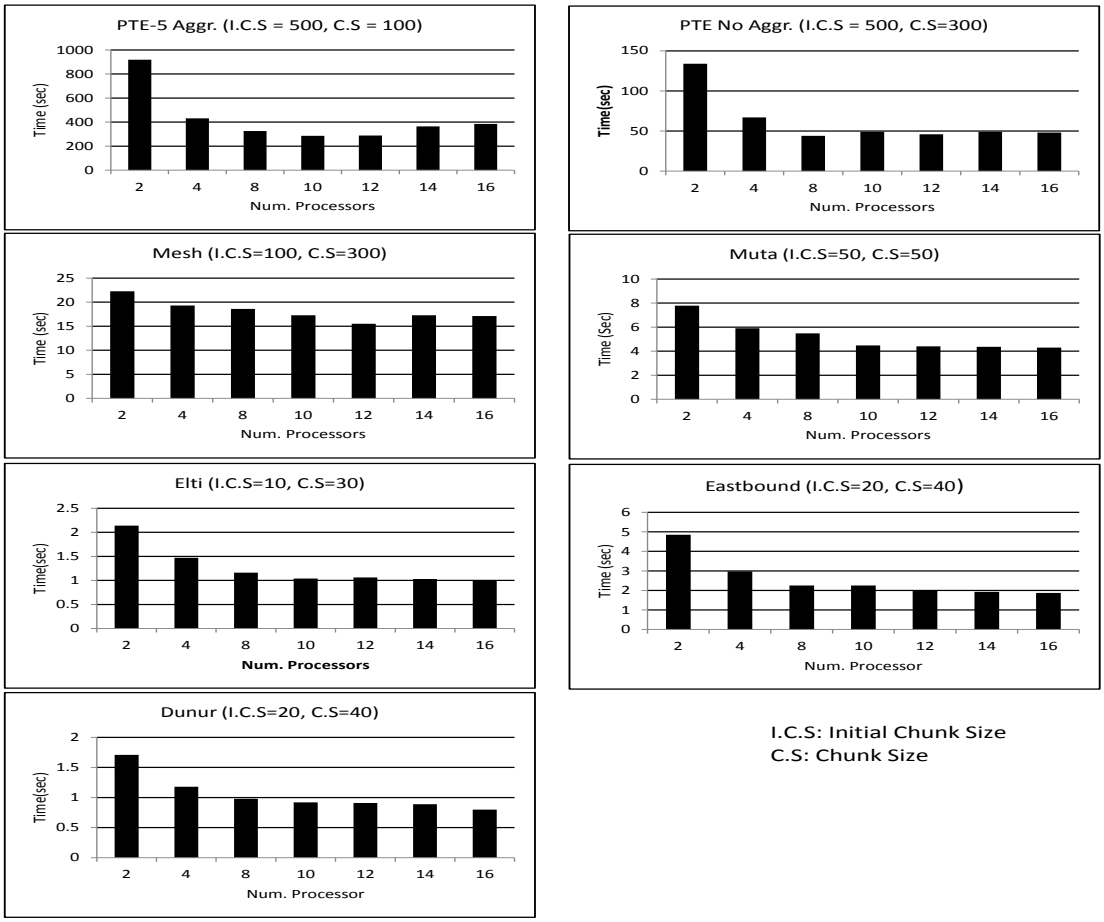
Data Set	**Search Space Evaluation and Pruning (mm:ss.s)**		**Search Space Formation (ss.s)**		**# Processors**	**Chunk Size**
	pCRIS	**Tabular CRIS**	**pCRIS**	**Tabular CRIS**		
Dunur	00:00.44	00:02.40	00.30	00.20	16	20
Elti	00:00.56	00:01.32	00.32	00.20	16	30
Eastbound	00:01.17	00:05.90	00.42	00.67	16	20
Mesh	00:09.80	00:24.00	02.00	02.21	12	100
Muta	00:03.76	00:11.20	01.71	01.80	4	50
PTE No Aggr	00:42.00	02:41.12	01.73	01.92	12	100
PTE 5 Aggr	04:44.03	35:01.00	21.00	28.00	8	50

Because the parallelization in the search space formation step does not lead to speedup in the rest of the experiments only the search space formation step is executed in parallel and the search space formation step is executed in sequential manner.

In order to analyze how pCRIS behaves with varying number of workers we conducted an other set of experiments. In Figure 6.2 we present the results. The experimental results show that there is a persistent increase in speedup up to 8 workers, but irregular after that point. We believe this is due to the MySQL's response to multiple connections to the same database instance.

In order to analyze the effect of the data chunk sizes on pCRIS's running time, we conducted two different experiments. In the first experiment, we set the minimum chunk size for parallel execution to 100 and run pCRIS five times for different initial chunk sizes. In the second experiment, initial chunk size was set to 200 and we changed the value of the minimum chunk size required for parallel execution. With the first experiment we aim to keep every worker busy until the very end of the execution and idle afterwards. In the second experiment we either allow work dispatching for very small sizes, say 1, or stop work dispatching at large data sizes, say 1000. The first experiment aims to allow busy communication at the beginning of the execution while the second experiment allows heavy communication through the end of the execution.

Figure 6.2: Average running times over five runs for varying number of workers



Experimental results show that pCRIS performs poorly with the extreme values. Setting the initial chunk size to a large value causes some fast workers to wait for the slow workers to finish their job before advancing to the next step. Setting the minimum chunk size to a very small value for parallel execution causes high communication cost and hence poor execution time. These observations indeed comply with Lowenthal's [96] study stating that a small block size decreases node idle time but increases the amount of communication, while a large block size decreases the amount of communication but increases node idle time and both situations resulting in efficiency loss. pCRIS tries to avoid falling in either situation by letting the user set the initial chunk size and minimum data size for parallel execution.

Figure 6.3 illustrates the speedup achieved for experiments with varying number of workers. The graph shows that pCRIS scales well with the number of SQL queries executed. For the PTE-5 Aggr experiment, pCRIS has a superlinear speedup [15], i.e. for four workers speedup is 6.2 and for 8 workers speedup is 9.3. As the number of queries executed drops, the performance of pCRIS worsens but still achieves some speedup.

Table6.24: Running Times for Different Chunk Sizes

Chunk Size	Running Time
20	8:32
100	5:14
200	4.42
500	4:59
700	5.04
1000	6:15

Table6.25: Running Times for Different Min. Chunk Sizes

Min. Chunk Size	Running Time
0	5.43
50	5.36
100	5.14
200	4.55
500	5.07
1000	5.20

Figure 6.4: Efficiency Results

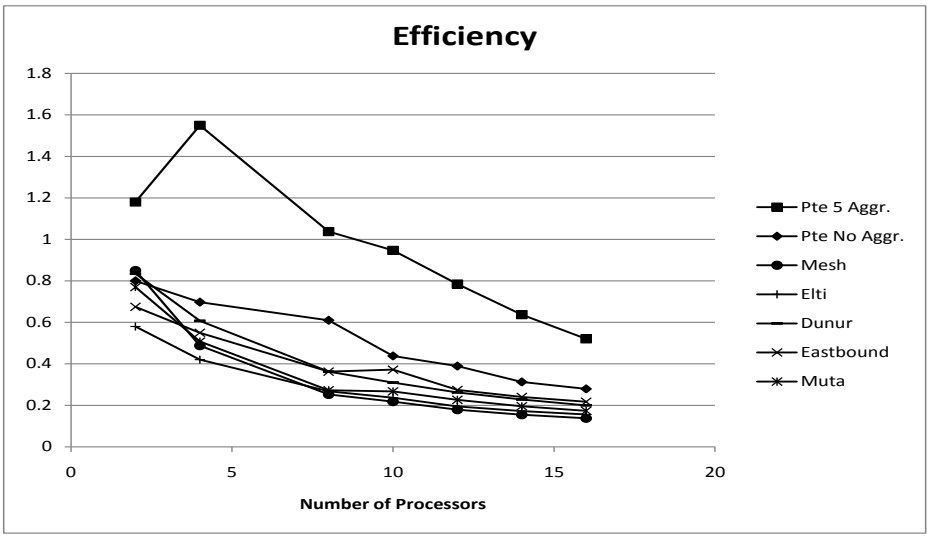
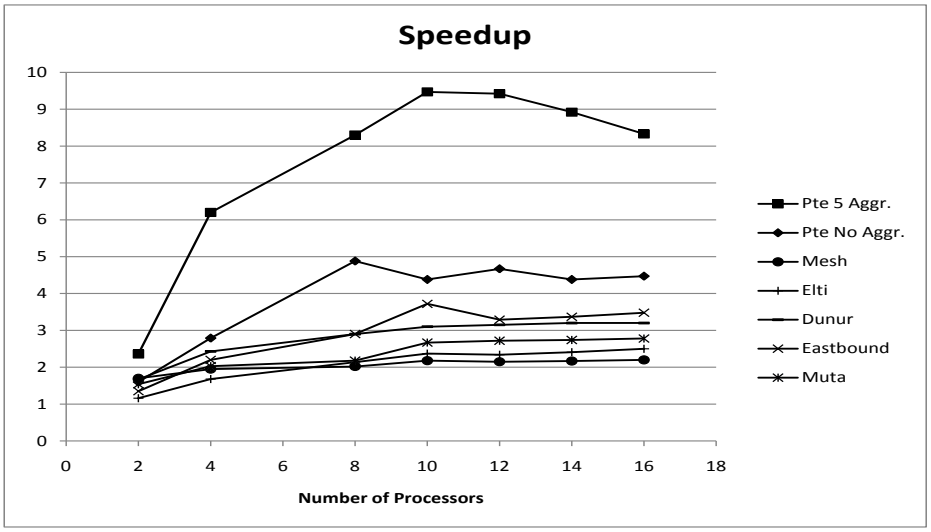


Figure 6.4 illustrates the efficiency of the workers in the system. Every experiment benefits until four workers, while experiments with larger search spaces still benefit from more workers. After eight workers, each experiment shows a sharp drop in efficiency measure, at the point when multiple connections are established to the same database instance. The efficiency measure dropping after eight workers suggests us that pCRIS may perform better if each worker has its own database instance.

Figure 6.3: Speedup Results

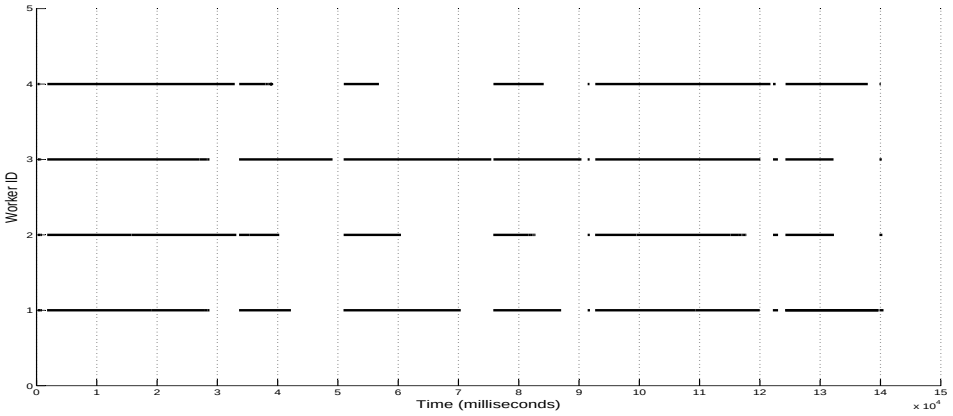


In Table 6.26 we compare the speedup of pCRIS with three other ILP-based parallel systems namely *pct*, *dplr*, and *dpilp* [63]. For PTE No Aggr test pCRIS outperforms the other three systems. For the Mesh data set *dpilp* performs better than pCRIS, while pCRIS outperforms the other two systems. Mesh is a sparse data set and has relatively small number of instances compared to PTE, PTE data set consists of 29267 instances of 32 relations and Mesh data set consist of 1749 instances of 26 relations. This imbalance in data sizes suggests that pCRIS is well suited for domains which have heavy SQL query executions.

Table6.26: Speedup Comparison

Data Set	**# Proc.**	**pCRIS**	**pct**	**dplr**	**dpilp**
PTE	2	2.36	0.75	1.34	1.20
	4	6.2	0.42	2.31	3.23
	8	8.3	-	0.79	5.17
Mesh	2	1.21	0.76	0.38	2.75
	4	1.53	0.79	0.53	3.46
	8	1.41	0.80	1.18	4.35
	16	1.34	-	1.21	4.37

Figure 6.5: Workers' Execution



We are unable to report the communication overhead, as data transfers are overlapping in most of the experiments and thereby hiding the exact cost. Figure 6.5 shows execution of four workers for the *PTE* data set with chunk size 500. Although every call to worker is not clear, the figure gives an idea on the overlapping processing. Indeed, for the test above, there are 18 calls for worker 1, 31 calls for worker 2, 16 calls for worker 3 and 24 calls to worker 4.

Similar to the statistical analysis of Selective Tabular CRIS, we employed Wilcoxon Mann Whitney test to analyze the statistical significance of the observed results.

p-values of the data sets for the Kolmogorov-Smirnov's test are listed in the first column of Table 6.27. With the risk factor $\alpha = 0.05$, each data set qualifies to be analyzed by the Wilcoxon-Mann-Whitney test. Results obtained for statistical significance are presented in Table 6.27. The statistical significance analysis is conducted with sample sets of 40 running times for each data set. Analysis results show that both the obtained median speedup is statistically significant.

Table6.27: Statistical significance of speedups

Data Set	**p Value**	**Speedup Median**	**Is Median Sign.**	**Median Conf Level**
Dunur	0.9945	1.48	True	0.99
Eastbound	0.81	3.847	True	0.99
Elti	0.4175	1.585	True	0.99
Mesh	0.787	1.353	True	0.99
Muta	0.9945	1.926	True	0.99
PTE No Aggr.	1.897	1.895	True	0.99
PTE 5 Aggr.	2.795	3.116	True	0.99

The speedup test tool also reports the overall gain, G, and overall speedup, S, for the the observations. When W is some weight function, G and S are defined as follows:

$$G = 1 - \frac{\sum_{j=1}^{b} W(C_j) \times ExecutionTime(C_j^{'}, I_j)}{\sum_{j=1}^{b} W(C_j) \times ExecutionTime(C_j, I_j)} \tag{6.4}$$

$$S = \frac{\sum_{j=1}^{b} W(C_j) \times ExecutionTime(C_j, I_j)}{\sum_{j=1}^{b} W(C_j) \times ExecutionTime(C_j^{'}, I_j)} \tag{6.5}$$

Table 6.28 reports overall speedup and gain for the experiments conducted on five processors. The reported confidence interval for the overall speedup and overall gain of the mean and the median is [0.561; 1].

Table6.28: Overall speedup and gain

	Min.	Mean	Median
Overall Speedup	3.77	3.93	3.84
Overall Gain	0.73	0.75	0.66

The test results confirm that pCRIS is consistent with CRIS in terms of rule coverage and accuracy since the concept induction mechanism remains unchanged. In Table 6.29 we report the coverage and accuracy results of pCRIS in comparison to that of similar systems. Note that for *Mesh* data set the result is for coverage, for the other data sets the result is for accuracy.

Table6.29: Coverage and Accuracy Results

System	**Mesh**	**Mutagenesis**	**PTE**
pCRIS	53 %	95%	86%
PosILP	42 %	90%	Not Avail.
SAHILP	38 %	89%	Not Avail.
PROGOL	31%	83%	72%
FOIL	31%	83%	Not Avail.

6.6 Evaluation on Graph-based Concept Discovery

In this section we firstly discuss the concept learning capability of the proposed approach and next its performance. In order to analyze its concept learning capability we performed experiments on data set with different characteristics.

In order to evaluate the performance and the accuracy of the proposed approach we conducted two sets of experiments. With the first set of the experiments we compare the proposed approach to the ILP-based concept discovery system CRIS. With the second set of the experiments we compare the proposed approach to graph based concept learning system Relational Paths-Based Learning.

The experiments are conducted on four different data sets, namely *Elti*, *Dunur*, *Same Genetation*, and *Family*. Although the data sets are very similar in nature, they are challenging in the concept discovery process due to the characteristics of the data. *Elti* is a real world kinship data set which contains transitive relations in the target concepts. *Dunur* is a real world kinship data set where facts indirectly

related to the target instances exist. The *Same-Gen* data set is also a real world kinship data set that contains recursive relations. *Family* [5] data set is highly relational data set that contains different kinship relations.

6.6.1 Learning Capability

To analyze the applicability of the proposed approach on data set that contain transitive facts we conducted experiments on the *Elti* data set. The proposed approach was successful in learning concept descriptors for such problems. The proposed approach outperformed CRIS in several aspects. Above all, the solution set it discovered does not contain any concept descriptors that are semantically identical but different in representation, i.e. differing in ordering of the literals or different renaming of the same literal argument. For the *Elti* data set CRIS discovered 12 concept descriptors while the proposed approach found 4 concept descriptors that have the same meaning.

To analyze the applicability of the proposed approach on data set which contains indirectly related facts we conducted experiments on the *Dunur* data set. The proposed approach was successful in discovering the concept descriptors in a domain where indirectly related fact were existent. It discovered the same set of the solution clauses as CRIS did.

To analyze the recursive rule learning capacity of the proposed approach we conducted a set of experiments on the *Same-Gen* data set. The proposed approach was successful in learning such theories and found the same set of solution clauses as CRIS did.

6.6.2 Performance Analysis

In this section we firstly compare the proposed approach to CRIS in terms of accuracy, coverage, and running time. For this purpose we conducted experiments on the *Elti*, *Dunur*, and *Same-Gen* data sets. In the second part we compare the proposed approach to RPBL in terms of number of solution clauses found, average length of the solution clauses, precision, and recall. For this comparison we conducted experiments on the *Family* data set.

In Table 6.30 we list the coverage and accuracy values of the proposed approach. In order to find the number of false positive and false negative instances, data sets are extended with their duals under the Close World Assumption. Coverage denotes the number of target instances of data set covered by the induced hypothesis set over the total number of target instances. Accuracy denotes the sum of correctly covered true positive and true negative instances over the sum of true positive, true negative, false positive and false negative instances.

Table6.30: Coverage and Accuracy Results

Data Set	Coverage	Accuracy
Elti	1.0	1.0
Dunur	1.0	1.0
Same-Gen	0.84	1.0

As the proposed approach discovered concept descriptors that cover the same set of target instances with CRIS, their accuracy and coverage values are identical.

Another problem with ILP-based concept discovery systems is their long running times. In Table 6.31 we compare running time of the proposed approach to CRIS. Experimental results show that the proposed approach has a shorter running time compared to CRIS. Although CRIS has powerful pruning strategies, it generalizes the concept descriptors in all possible ways which results in a large search space. On the other hand, the proposed approach limits it search space by possible graph expansions only and has a relatively small search space.

Table6.31: Running Times in Seconds

Data Set	The Proposed Approach	CRIS
Elti	0.51	35
Dunur	0.11	26
Same-Gen	0.76	12

ILP-based concept discovery systems suffer from long execution times as such systems usually execute huge number of queries to evaluate the search space. Execution of each such query may be costly especially for long concept descriptors as the execution of a query requires as many table joins as the length of the concept descriptor. In Table 6.32 we compare the number of queries executed for pruning purposes.

Table6.32: Queries

Data Set	The Proposed Approach	CRIS
Elti	2118	3571
Dunur	3192	3777
Same-Gen	222	996

As the results show, the proposed approach executes less number of queries compared to CRIS. Drop in the number of executed queries is especially apparent for the *Elti* and *Same-Gen* data sets. For these data sets the proposed approach finds solution sets with less number of concept descriptors which have the same semantical meaning with that of CRIS.

The experimental results discussed above show that the proposed approach is comparable and compatible to ILP-based concept discovery system called CRIS.

To compare the proposed approach to RPBL we conducted a set of experiments on the *Family* data set. We set the minimum support to 0.1, minimum confidence to 0.7 and maximum rule length according to the length of the longest path found by RPBL. In Table 6.33[1] we list the observed results. The third column lists the number of the solution clauses induced, the fourth column lists the average length of solution clauses. The last two columns list the precision and recall values. We did not run RPBL on our machines but retrieved the results from [65].

As the experimental results show the proposed approach finds larger number of concept descriptors with less number of literals. Concept descriptors with many literals may be hard to interpret and such concept descriptors are subject to the overfitting problem. The proposed approach is capable of inducing much simpler concept descriptors with almost the same precision and recall values. The proposed approach missed to define 1 target instance for the husband, nephew, and wife relations; 2

[1] Some of the induced solution clauses are of length 2, and some are of length 3. Their average is not an integer value.

Table6.33: Family data set results

Target	Algorithm	#Clauses	Length	Precision	Recall
Brother	Proposed	2	2.5	100	100
	RPBL	2	6	95	96
Uncle	Proposed	9	2	100	100
	RPBL	2	6	100	100
Niece	Proposed	7	2	100	98
	RPBL	2	6	100	100
Aunt	Proposed	10	2	100	100
	RPBL	2	6	100	100
Nephew	Proposed	7	2	100	99
	RPBL	2	6	100	100
Son	Proposed	3	2	100	94
	RPBL	2	6	100	100
Mother	Proposed	6	2	100	100
	RPBL	2	6	100	100
Father	Proposed	6	2	100	100
	RPBL	2	6	100	100
Daughter	Proposed	3	2	100	96
	RPBL	2	6	100	100
Sister	Proposed	4	2.7	100	100
	RPBL	2	6	96	99
Wife	Proposed	1	2	100	96
	RPBL	1	3	100	100
Husband	Proposed	1	2	100	96
	RPBL	1	3	100	100

target instances for the niece and daughter relations; and 4 target instances for the son relation. Indeed these misses are due to the minimum support and confidence values. For example, for the wife relations there is 1 uncovered target instance. As 1 / 25 is less than the minimum support value, the proposed approach does not attempt to find a concept descriptor for the uncovered target instance. Average concept descriptor learning time for RPBL is around 0.02, and it is around 1 second for the proposed approach.

Similar to the run of RPBL on the *Family* data set, we did not include the negative target instances in the concept learning process but employed CWA.

CHAPTER 7

CONCLUSION

Increasing amount of the data stored on relational databases has triggered the development of mining algorithms that can directly work on such data. Inductive Logic Programming (ILP) is one the most successful attempts to handle such data.

ILP systems employ first order logic as the representation framework for data and employ various logic based techniques to discover the hidden patterns. The induced patterns are also represented within the first order logic framework, generally as Horn clauses.

Based on the pattern such systems discover, they are classified as descriptive systems or predictive systems. In descriptive ILP, the systems looks for any valuable pattern in the data. In the descriptive ILP, however, the aim is to find patterns that define some target concept.

Predictive ILP, also referred *concept discovery*, has widely been applied in various domains and successful experimental results have been reported. Concept discovery systems input a finite set of target instances, a finite set of background facts, and some parameters to constrain the properties of the induced concept descriptors. Concept discovery can be considered as a supervised learning as the target instances are labeled as *positive* if they belong to the target relation, or *negative* if they do no. Background data consists of a set of fact which are directly or indirectly related to the target instances. Such systems also require some user defined parameters to constrain the search space and evaluate the quality of concept descriptors.

One of the problems concept discovery systems face is scalability and efficiency. Such systems usually experience scalability issues, both execution time wise and memory wise, as they create a large search spaces and evaluation of the search space is a costly process. Among others, methods based on query transformation, lazy evaluation, and introduction of language bias have been introduced to improve scalability of concept discovery systems.

In this study we focus on the scalability issue of concept discovery systems. We attack this from two seemingly different points. First attempt is based on employing memoization and the second one is based on parallelization. The proposed methods are implemented as extension to CRIS, which is an ILP-based predictive learning system.

Memoization has extensively been applied in problems to improve running time. This technique improves the running the time of systems if they make calls to the same functions with the same parameters. In memoization, results of functions are stored in look-up tables and a solution of a function is directly retrieved from the look-up table when it is re-executed with the previously called parameters. Memoization is well applicable to concept discovery systems as such systems frequently generate repeating executing queries.

To improve efficiency of concept discovery systems we developed three methods. The first method, called Tabular CRIS, aims to catch the repeating queries that are generated within the same epoch. To realize this, Tabular CRIS maintains a look-up to store the evaluation queries. Tabular CRIS sends an evaluation query to the database engine if it is not executed before. If a query is executed before its result is retrieved from the look-up table. Tabular CRIS stored the evaluation query and the number of tuples returned by the query in the look-up tables.

In case of noisy and incomplete data, concept discovery systems usually runs multiple iterations on the data set. At each iteration some hypotheses that explain a subset of the target instances are found, and those explained target instances are removed from the data set. A new iteration is performed to find concept descriptors that explain the remaining target instances. In such cases, an evaluation query that is generated in the current iteration may have been generated and evaluated in one of the previous iterations. Tabular CRIS is not able to handle such repeating queries as it cleans the content of the look-up tables at the end of each iteration. In order to cope with such repeating queries generated at different iterations we propose Tabular CRIS-wEF. Different than Tabular CRIS, it stores the evaluation query and the tuples returned by the evaluation query in the look-up table. As removing the target instances that are explained by some concept descriptors may change the result of the evaluation queries stored in the look-up tables, the covering function is modified to remove the explained target instances also from the look-up tables. With this modification on the covering function, look-up tables always store the updated resultsets of the queries.

Although Tabular CRIS-wEF improved hash look-up table hit count of Tabular CRIS in considerable extent, it requires relatively more memory to main the look-up table. Although it is not possible to figure out if a query will be regenerated in the following iterations, in some cases it is possible to decide that a certain query will not be. Tabular CRIS-S is proposed to deal the storage of such non-repetitive evaluation queries. Tabular CRIS-S implements policies on what kind of queries to store in the look-up tables and for how long.

Those proposed methods differ from the state of the art memoization based concept discovery systems by focusing on the evaluation queries instead of concept descriptors. Considering evaluation queries instead of concept descriptors itself has a major contribution as different concept descriptors may map into same evaluation query. Also, the proposed methods are implemented within the concept discovery tool, which makes it transparent to the under laying data storage engine. Several studies proposed in the literature modify the underlaying data engine, which is Prolog in most cases, to handle such memoization issues.

Experimental results of Tabular CRIS, Tabular CRIS-wEF, and Tabular CRIS-S have proved that memoization has greatly improved the running time of CRIS with affordable memory requirements. When compared to stat of the art systems, the proposed approaches require less memory and gain higher speedups.

ILP-based concept discovery systems are well amenable for parallelization as they perform large number of reads on the data set to evaluate the concept descriptors. Considering this we propose a parallel version of CRIS, called pCRIS. pCRIS employs implements master-worker architecture in shared nothing environment and realizes data-parallelization paradigm. In pCRIS there is master node which generates the job and patches it among multiple workers for evaluation. As it works on a shared nothing environment data must be made available to each worker beforehand. pCRIS is parametric as it requires user inputs to decide which parts to run in parallel, minimum workload to run a function in parallel.

pCRIS parallelizes both the search space construction and evaluation steps of CRIS. To parallelize the

search space formation step, the master splits the current search space among workers in such a way that each worker refines a mutually exclusive subset of the current search space, and sends the result back to the master where the global search space for the next iteration is formed.

To parallelize the search space evaluation step, the master node maps the concept descriptors into evaluation queries and dispatches them among workers. The master node also maintains a look-up table to store the evaluation queries sent to workers to avoid re-executing of the regenerated queries.

pCRIS is designed in such a way that, based on the workload, it may start execution of function in parallel and when the workload drops below a user defined threshold suspends the workers and serially executes the code for the remaining workload.

Experimental results show that pCRIS well scales with the size of the data. We have also observed that speedup is more significant for the experiments whose search space and the data set are large. Even for the PTE-5 Aggr. data set super linear speedup is achieved. Also when compared to state of the art systems pCRIS achieved higher speedups.

Graph based approaches are recently attracting more attention from concept discovery community. Graph based approaches either look for frequently appearing substructures or for finite length paths that connect similar objects. Such approaches require costly computations to compare the similarity of the substructures and complex indexing mechanisms to keep track of the paths. In this study we propose a new method to represent the data on the graph. We utilize directed multigraph as the representation framework. Instead of representing each fact as distinct vertices in the graph, we propose to represent similar facts as a single vertex. By this way the representation of the data becomes more compact and easier for human interpretation. We consider two facts similar if they are related to a common fact with the same relation. We label the edges after the relation names that holds between related vertices and each child vertex stores the path from the root to itself. By this way, while constructing the graph we also form the possible concept descriptors. Once the graph is constructed, the path stored at the leaf vertices are translated into SQL queries and they are sent to database server for evaluation. Initial experimental results show that the proposed approach is capable of representing the data in less number of concept descriptors with almost the same accuracy values.

Some of the open problems related to the methods studied in his dissertation are listed below:

- Effect of different hashing functions on speedup in memoization based approaches need to be analyzed.
- The chunk size calculation mechanism of pCRIS may even be improved by employing prior statistical information on the distribution of data.
- The current implementation of the graph based approach is limited with the binary relations. This need to be extended to n-ary relations.
- Data may be preprocessed to find closely related facts and the graph based algorithm can be parallelized in such a way that each worker executes on such more related sub-data set.

REFERENCES

[1] http://www.ceng.metu.edu.tr/hpc/index/. [Last accessed December, 2012].

[2] http://dev.mysql.com/doc/refman/5.0/en/index.html. [Last accessed December, 2012].

[3] http://www.boost.org/users/history/version_1_37_0.html. [Last accessed December, 2012].

[4] DCC-2003-03 (2003) on the implementation of an ILP system with prolog. Technical report, Laboratorio de Inteligencia Artificial e Ciencia de Computadores, Porto, Porugal.

[5] Large family dataset. http://www.cs.utexas.edu/ftp/mooney/forte/. Online; accessed December 09, 2012.

[6] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/. Last accessed December, 2012.

[7] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/. Last accessed December, 2012.

[8] PRG-TR-25-96 (1996) Part-of-speech disambiguation using ilp. Technical report, Oxford University Computing Laboratory, Oxford, UK.

[9] PRG-TR-8-95 (1995) Theories for mutagenicity: A study of first-order and feature based induction. Technical report, Oxford University Computing Laboratory, Oxford, UK.

[10] Using Microsoft Message Passing Interface. http://technet.microsoft.com/en-us/library/4cb68e33-024b-4677-af36-28a1ebe9368f. Last accessed December, 2012.

[11] H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[12] 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.

[13] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8:962–969, December 1996.

[14] K. Aida, W. Natsume, and Y. Futakata. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In *CCGrid 2003: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 156 – 163, May 2003.

[15] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

[16] É. Alphonse, T. Girschick, F. Buchwald, and S. Kramer. A numerical refinement operator based on multi-instance learning. In P. Frasconi and F. A. Lisi, editors, *ILP*, volume 6489 of *Lecture Notes in Computer Science*, pages 14–21. Springer, 2010.

[17] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Spring Joint Computing Conference*, volume 30 of *AFIPS Conference Proceedings*, pages 483–485. AFIPS / ACM / Thomson Book Company, Washington D.C., 1967.

[18] M. S. Amin, R. L. F. Jr., and H. M. Jamil. Top-k similar graph matching using tram in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(6):1790–1804, 2012.

[19] C. Anglano, A. Giordana, G. L. Bello, and L. Saitta. An experimental evaluation of coevolutive concept learning. In J. W. Shavlik, editor, *ICML*, pages 19–27. Morgan Kaufmann, 1998.

[20] E. Bauer and G. Kókai. Learning from noise data with the help of logic programming systems. In *Proceedings of the 4th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering Data Bases*, pages 32:1–32:6, Salzburg, Austria, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[21] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.

[22] H. Blockeel, S. Dzeroski, and J. Grbovic. Simultaneous prediction of mulriple chemical parameters of river water quality with tilde. In J. M. Zytkow and J. Rauch, editors, *PKDD*, volume 1704 of *Lecture Notes in Computer Science*, pages 32–40. Springer, 1999.

[23] H. Blockeel and L. D. Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

[24] H. Blockeel, L. D. Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Min. Knowl. Discov.*, 3(1):59–93, 1999.

[25] H. Blockeel and M. Sebag. Scalability and efficiency in multi-relational data mining. *SIGKDD Explorations*, 5(1):17–30, 2003.

[26] H. Blockeel, T. Witsenburg, and J. N. Kok. Graphs, hypergraphs, and inductive logic programming. In *MLG*, 2007.

[27] F. Bonchi, C. Castillo, A. Gionis, and A. Jaimes. Social network analysis and mining for business applications. *ACM Trans. Intell. Syst. Technol.*, 2(3):22:1–22:37, May 2011.

[28] M. Botta. Challenging relational learning problems. `http://www.di.unito.it/~mluser/challenge/index.html`, 2004. Online; accessed July 29, 2012.

[29] M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning: Hard problems and phase transitions. In *Proceedings of the Advances in Artificial Intelligence, 6th Congress of the Italian Association for Artificial Intelligence*, pages 178–189, Bologna, Italy, September 14-17 1999.

[30] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[31] R. Camacho. Indlog – induction in logic. In J. J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence*, volume 3229 of *Lecture Notes in Computer Science*, pages 718–721. Springer Berlin Heidelberg, 2004.

[32] R. Camacho, N. A. Fonseca, R. Rocha, and V. S. Costa. Ilp : - just trie it. In H. Blockeel, J. Ramon, J. W. Shavlik, and P. Tadepalli, editors, *ILP*, volume 4894 of *Lecture Notes in Computer Science*, pages 78–87. Springer, 2007.

[33] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge-elicitation tool for sophisticated users. In *Proceedings of 2nd European Working Session on Learning*, pages 31–45, Bled, Yugoslavia, May 1987.

[34] G. Chartrand. *Introductory Graph Theory*. Dover Publications, 1984.

[35] J. Cheng. Dependence analysis of parallel and distributed programs and its applications. In *Advances in Parallel and Distributed Computing, 1997. Proceedings*, pages 370–377, March 1997.

[36] A. Clare and R. D. King. Predicting gene function in saccharomyces cerevisiae. In *ECCB*, pages 42–49, 2003.

[37] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, 1998.

[38] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res. (JAIR)*, 1:231–255, 1994.

[39] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[40] V. S. Costa, A. Srinivasan, and R. Camacho. A note on two simple transformations for improving the efficiency of an ilp system. In J. Cussens and A. M. Frisch, editors, *ILP*, volume 1866 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2000.

[41] V. S. Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W. V. Laer. Query transformations for improving the efficiency of ilp systems. *Journal of Machine Learning Research*, 4:465–491, 2003.

[42] J. Cussens. Bayes and pseudo-bayes estimates of conditional probabilities and their reliability. In P. Brazdil, editor, *ECML*, volume 667 of *Lecture Notes in Computer Science*, pages 136–152. Springer, 1993.

[43] M. Davis, W. Liu, P. Miller, and G. Redpath. Detecting anomalies in graphs with numeric labels. In *CIKM*, pages 1197–1202, 2011.

[44] 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. Springer-Verlag, 1997.

[45] L. Dehaspe and L. D. Raedt. Parallel inductive logic programming. In *In Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 112–117, 1995.

[46] L. Dehaspe and H. Toironen. Relational data mining. chapter Discovery of relational association rules, pages 189–208. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[47] F. DiMaio and J. W. Shavlik. Learning an approximation to inductive logic programming clause evaluation. In R. Camacho, R. D. King, and A. Srinivasan, editors, *ILP*, volume 3194 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2004.

[48] B. Dolsak. Finite element mesh design expert system. *Knowl.-Based Syst.*, 15(8):315–322, 2002.

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

[50] A. Doncescu, J. Waissman, G. Richard, and G. Roux. Characterization of bio-chemical signals by inductive logic programming. *Knowl.-Based Syst.*, 15(1-2):129–137, 2002.

[51] Y. Dong, X. Du, Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. Smolka, O. Sokolsky, E. Stark, and D. Warren. Fighting livelock in the i-Protocol: A comparative study of verification tools. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 1579, pages 74–88, Amsterdam, The Netherlands, March 22-28 1999. Springer Berlin / Heidelberg.

[52] S. Dudoit, J. P. Shaffer, and J. C. Boldrick. Multiple hypothesis testing in microarray experiments. *Statistical Science*, 18(1):71–103, 2003.

[53] S. Džeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the 4th Scandinavian Conference on Artificial intelligence*, pages 111–125, Stockholm, Sweden, May 4-7 1993. IOS Press, Amsterdam, The Netherlands.

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

[55] S. Dzeroski, N. Jacobs, M. Molina, C. Moure, S. Muggleton, and W. V. Laer. Detecting traffic problems with ilp. In Page [121], pages 281–290.

[56] S. Dzeroski, S. Schulze-Kremer, K. Heidtke, K. Siems, and D. Wettschereck. Applying ilp to diterpene structure elucidation from 13c nmr spectra. In S. Muggleton, editor, *Inductive Logic Programming*, volume 1314 of *Lecture Notes in Computer Science*, pages 41–54. Springer Berlin Heidelberg, 1997.

[57] D. Eager, J. Zahorjan, and E. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408 423, March 1989.

[58] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In L. Birnbaum and G. Collins, editors, *ML*, pages 403–406. Morgan Kaufmann, 1991.

[59] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901 – 1909, December 1966.

[60] N. Fonseca, R. Rocha, R. Camacho, and F. Silva. Efficient data structures for inductive logic programming. In *Proceedings of the 13th International Conference Inductive Logic Programming*, pages 130–145, Szeged, Hungary, September 29-October 1 2003. Springer-Verlag.

[61] N. Fonseca, F. Silva, and R. Camacho. April An inductive logic programming system. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence*, pages 481–484. Springer Berlin / Heidelberg, Liverpool, UK, September 13-15 2006.

[62] N. A. Fonseca, F. M. A. Silva, V. S. Costa, and R. Camacho. A pipelined data-parallel algorithm for ILP. In *CLUSTER 2005: IEEE International Conference on Cluster Computing*, pages 1–10, 2005.

[63] N. A. Fonseca, O. Silva, and R. Camacho. Strategies to parallelize ILP systems. In *ILP 2005: Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 3625 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[64] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[65] Z. Gao, Z. Zhang, and Z. Huang. Extensions to the relational paths based learning approach rpbl. In N. T. Nguyen, H. P. Nguyen, and A. Grzech, editors, *ACIIDS*, pages 214–219. IEEE Computer Society, 2009.

[66] Z. Gao, Z. Zhang, and Z. Huang. Learning relations by path finding and simultaneous covering. In *CSIE (5)*, pages 539–543, 2009.

[67] F. Giannotti, G. Manco, and J. Wijsen. Logical languages for data mining. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 325–361. Springer, 2003.

[68] A. Giordana, L. Saitta, M. Sebag, and M. Botta. Analyzing relational learning in the phase transition framework. In *ICML*, pages 311–318, Stanford, CA, USA, June 29 - July 2 2000.

[69] J. A. Gonzalez, L. B. Holder, and D. J. Cook. Graph based concept learning. In H. A. Kautz and B. W. Porter, editors, *AAAI/IAAI*, page 1072. AAAI Press / The MIT Press, 2000.

[70] J. A. Gonzalez, L. B. Holder, and D. J. Cook. Graph-based relational concept learning. In *ICML*, pages 219–226, 2002.

[71] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *ECIR'05: Proceedings of the 27th European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[72] G. Graefe and W. McKenna. The Volcano optimizer generator: extensibility and efficient search. In *Proceedings of the 9th International Conference on Data Engineering*, pages 209–218, Vienna, Austria, April 19-23 1993. IEEE.

[73] M. Hall and J. P. McNamee. Improving software performance with automatic memoization. *Johns Hopkins APL Technical Digest*, 18(2), 1997.

[74] E. H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):377–352, May/June 2000.

[75] F. Harary and R. Read. Is the null-graph a pointless concept? In R. Bari and F. Harary, editors, *Graphs and Combinatorics*, volume 406 of *Lecture Notes in Mathematics*, pages 37–44. Springer Berlin Heidelberg, 1974.

[76] J. Hekanaho. Dogma: A ga-based relational learner. In Page [121], pages 205–214.

[77] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In *GRID '00: Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing*, pages 214–227. Springer-Verlag, 2000.

[78] L. B. Holder, D. J. Cook, and S. Djoko. Substucture discovery in the subdue system. In U. M. Fayyad and R. Uthurusamy, editors, *KDD Workshop*, pages 169–180. AAAI Press, 1994.

[79] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods, Edition 2*. Wiley-Interscience, 1999.

[80] E. E. Johnson. Completing an mimd multiprocessor taxonomy. *SIGARCH Comput. Archit. News*, 16(3):44–47, June 1988.

[81] Y. Kavurucu. *ILP-based Concept Discovery Method for Multi-relational Data Mining*. PhD thesis, Middle East Technical University, 2009.

[82] Y. Kavurucu, P. Senkul, and I. Toroslu. A comparative study on ilp-based concept discovery systems. *Expert Systems with Applications*, 38(9):11598–11607, 2011.

[83] Y. Kavurucu, P. Senkul, and I. H. Toroslu. Concept discovery on relational databases: New techniques for search space pruning and rule quality improvement. *Knowl.-Based Syst.*, 23(8):743–756, 2010.

[84] R. King, A. Srinivasan, and L. Dehaspe. Warmr: a data mining tool for chemical data. *Journal of Computer-Aided Molecular Design*, 15:173–181, 2001.

[85] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[86] S. Konstantopoulos. A data-parallel version of Aleph. *CoRR*, abs/0708.1527, 2007.

[87] S. Kramer, N. Lavrac, and P. Flach. *Propositionalization Approaches to Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.

[88] M.-A. Krogel, S. Rawles, F. Zelezný, P. A. Flach, N. Lavrac, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In T. Horváth, editor, *ILP*, volume 2835 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2003.

[89] M.-A. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, pages 142–155, Strasbourg, France, September 9-11 2001. Springer-Verlag.

[90] R. Kufrin. Generating C4.5 production rules in parallel. In *AAAI-1997: Proceedings of the 14th National Conference on Artificial Intelligence*, pages 565–570, 1997.

[91] N. Kurumatani, H. Monji, and T. Ohkawa. Binding site extraction by similar subgraphs mining from protein molecular surfaces. In *Bioinformatics Bioengineering (BIBE), 2012 IEEE 12th International Conference on*, pages 255–259, nov. 2012.

[92] W. V. Laer, L. Dehaspe, and L. D. Raedt. Applications of a logical discovery engine. In *KDD Workshop*, pages 263–274, 1994.

[93] W. V. Laer and L. D. Raedt. How to upgrade propositional learners to first order logic: A case study. In G. Paliouras, V. Karkaletsis, and C. D. Spyropoulos, editors, *Machine Learning and Its Applications*, volume 2049 of *Lecture Notes in Computer Science*, pages 102–126. Springer, 2001.

[94] N. Lavrac, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with linus. In *Proceedings of the 6th European Working Session on Machine Learning*, pages 265–281, Porto, Portugal, March 6-8 1991. Springer-Verlag.

[95] H. Lodhi, S. Muggleton, and M. J. Sternberg. Multi-class protein fold recognition using large margin logic based divide and conquer learning. In *Proceedings of the KDD-09 Workshop on Statistical and Relational Learning in Bioinformatics*, pages 22–26, Paris, France, June 28 2009. ACM.

[96] D. K. Lowenthal. Accurately selecting block size at runtime in pipelined parallel programs. *International Journal of Parallel Programming*, 28:245–274, June 2000.

[97] T. Matsuda, H. Motoda, T. Yoshida, and T. Washio. Knowledge discovery from structured data by beam-wise graph-based induction. In M. Ishizuka and A. Sattar, editors, *PRICAI*, volume 2417 of *Lecture Notes in Computer Science*, pages 255–264. Springer, 2002.

[98] T. Matsui, N. Inuzuka, H. SEKI, and H. Itoh. Comparison of three parallel implementations of an induction algorithm. In *In 8th Int. Parallel Computing Workshop*, pages 181–188, Singapore, September 1998. Springer-Verlag.

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

[100] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence"*, 20(2):111–161, 1983.

[101] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. Dynamic join optimization in multi-hop wireless sensor networks. *PVLDB*, 3(1):1279–1290, 2010.

[102] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, USA, 1997.

[103] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *J. Artif. Intell. Res. (JAIR)*, 8:67–91, 1998.

[104] J.-I. Motoyama, S. Urazawa, T. Nakano, and N. Inuzuka. A mining algorithm using property items extracted from sampled examples. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming*, volume 4455 of *Lecture Notes in Computer Science*, pages 335–350. Springer Berlin Heidelberg, 2007.

[105] I. Mozetic. Newgem: Program for learning from examples, technical documentation and user's guide. In *Reports of Intelligent Systems Group UIUCDCSF-85-949, Department of Computer Science, University of Illinois, Urbana Champaign, IL,*, 1985.

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

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

[108] S. Muggleton. Inductive logic programming: Issues, results and the challenge of learning language in logic. *Artif. Intell.*, 114(1-2):283–296, 1999.

[109] S. Muggleton and W. L. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352, Ann Arbor, Michigan, USA, June 12-14 1988.

[110] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381, Tokyo, Japan, October 8-10 1990.

[111] S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic. In *International Workshop on Inductive Logic Programming*, 1992.

[112] A. Mutlu, M. A. Berk, and P. Senkul. Improving the time efficiency of ilp-based multi-relational concept discovery with dynamic programming approach. In E. Gelenbe, R. Lent, G. Sakellari, A. Sacan, I. H. Toroslu, and A. Yazici, editors, *ISCIS*, volume 62 of *Lecture Notes in Electrical Engineering*, pages 373–376. Springer, 2010.

[113] A. Mutlu and P. Senkul. Improving hit ratio of ilp-based concept discovery system with memoization, 2012. Accepted for publication in the Computer Journal, DOI: 10.1093/comjnl/bxs163.

[114] A. Mutlu and P. Senkul. Selective memoization for ilp-based concept discovery systems, 2012. Submitted to Knowledge and Information Systems.

[115] A. Mutlu and P. Senkul. Improving hash table hit ratio of an ilp-based concept discovery system with memoization capabilities. In E. Gelenbe and R. Lent, editors, *Computer and Information Sciences III*, pages 261–269. Springer London, 2013.

[116] H. Nassif, H. Al-Ali, S. Khuri, W. Keirouz, and D. Page. An inductive logic programming approach to validate hexose binding biochemical knowledge. In *Proceedings of the 19th International Conference on Inductive Logic Programming*, pages 149–165, Leuven, Belgium, July 2-4 2010. Springer.

[117] C. Nattee, S. Sinthupinyo, M. Numao, and T. Okada. Learning first-order rules from data with multiple parts: applications on mining chemical compound data. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 77–, New York, NY, USA, 2004. ACM.

[118] A. Nica. A call for order in search space generation process of query optimization. In *27th International Conference on Data Engineering Workshops*, pages 4–9, Hannover, Germany, April 11-16 2011. IEEE.

[119] H. Ohwada, H. Nishiyama, and F. Mizoguchi. Concurrent execution of optimal hypothesis search for inverse entailment. In *ILP'00: Proceedings of 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Computer Science*, pages 165–173. Springer Berlin / Heidelberg, 2000.

[120] I. M. Ong, I. de Castro Dutra, D. Page, and V. S. Costa. Mode directed path finding. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 673–681. Springer, 2005.

[121] D. Page, editor. *Inductive Logic Programming, 8th International Workshop, ILP-98, Madison, Wisconsin, USA, July 22-24, 1998, Proceedings*, volume 1446 of *Lecture Notes in Computer Science*. Springer, 1998.

[122] J. S. Park, M.-S. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In *CIKM '95: Proceedings of the 4th International Conference on Information and Knowledge Management*, pages 31–36. ACM, 1995.

[123] M. J. Pazzani, C. Brunk, and G. Silverstein. A knowledge-intensive approach to learning relational concepts. In *ML*, pages 432–436, 1991.

[124] G. Penn and C. Munteanu. A tabulation-based parsing method that reduces copying. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 200–207, Sapporo, Japan, July 7-12 2003. ACL.

[125] N. L. Peter, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *Proceedings of the 9th International Workshop on Inductive Logic Programming*, pages 174–185, Bled, Slovenia, June 24-27 1999. Springer-Verlag.

[126] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

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

[128] J. R. Quinlan. Determinate literals in inductive logic programming. In *IJCAI*, pages 746–750, 1991.

[129] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[130] L. D. Raedt and M. Bruynooghe. A theory of clausal discovery. In R. Bajcsy, editor, *IJCAI*, pages 1058–1063. Morgan Kaufmann, 1993.

[131] R. Reiter. On closed world data bases. In *Proceedings of the Symposium on Logic and Data Bases*, pages 55–76, Toulouse, France, 1977. Plemum Press, New York.

[132] A. Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335. Springer Berlin Heidelberg, 2004.

[133] B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *AAAI*, pages 50–55, 1992.

[134] J. Rissanen. *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1989.

[135] M. Robnik-Sikonja and I. Kononenko. An adaptation of relief for attribute estimation in regression. In D. H. Fisher, editor, *ICML*, pages 296–304. Morgan Kaufmann, 1997.

[136] R. Rocha. On improving the efficiency and robustness of table storage mechanisms for tabled evaluation. In *PADL*, pages 155–169, 2007.

[137] R. Rocha, N. A. Fonseca, and V. S. Costa. On applying tabling to inductive logic programming. In *Proceeedings of the 16th European Conference on Machine Learning*, pages 707–714, Porto, Portugal, October 3-7 2005. Springer.

[138] C. J. Romanowski, R. Nagi, and M. Sudit. Data mining in an engineering design environment: Or applications from graph matching. *Computers & Operations Research*, 33(11):3150 – 3160, 2006.

[139] O. E. Romero, J. A. Gonzalez, and L. B. Holder. Handling of numeric ranges for graph-based knowledge discovery. In H. W. Guesgen and R. C. Murray, editors, *FLAIRS Conference*. AAAI Press, 2010.

[140] O. E. Romero, J. A. Gonzalez, and L. B. Holder. Handling of numeric ranges with the subdue system. In *FLAIRS Conference*, 2011.

[141] K. F. Sagonas and P. J. Stuckey. Just enough tabling. In E. Moggi and D. S. Warren, editors, *PPDP*, pages 78–89. ACM, 2004.

[142] J. C. A. Santos, A. Tamaddoni-Nezhad, and S. Muggleton. An ilp system for learning head output connected predicates. In *EPIA*, pages 150–159, 2009.

[143] M. Sebag and C. Rouveirol. Any-time relational reasoning: Resource-bounded induction and deduction through stochastic matching. *Machine Learning*, 38(1-2):41–62, 2000.

[144] M. Serrurier and H. Prade. Improving inductive logic programming by using simulated annealing. *Information Sciences*, 178(6):1423–1441, 2008.

[145] E. Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA, 1983.

[146] H. J. Siegel and S. Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. *Journal of Systems Architecture*, 46(8):627 – 639, 2000.

[147] J. Sowa. Conceptual graph summary. In *Conceptual Structures: Current Research and Practice*, pages 3–66, 1992.

[148] L. Specia, M. Stevenson, and M. das Graças Volpe Nunes. Learning expressive models for word sense disambiguation. In J. A. Carroll, A. van den Bosch, and A. Zaenen, editors, *ACL*. The Association for Computational Linguistics, 2007.

[149] T. P. Speed. Review of 'stochastic complexity in statistical inquiry' (rissanen, j.; 1989). *IEEE Transactions on Information Theory*, 37(6):1739–, 1991.

[150] A. Srinivasan. The Aleph Manual. `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/`, 1999. Online; accessed July 29, 2012.

[151] A. Srinivasan. A study of two sampling methods for analyzing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

[152] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1–6. Morgan Kaufmann, Stockholm, Sweden, July 31-August 6 1997.

[153] A. Srinivasan and R. Kothari. A study of applying dimensionality reduction to restrict the size of a hypothesis space. In S. Kramer and B. Pfahringer, editors, *ILP*, volume 3625 of *Lecture Notes in Computer Science*, pages 348–365. Springer, 2005.

[154] J. Struyf and H. Blockeel. Query optimization in inductive logic programming by reordering literals. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, pages 329–346, Szeged, Hungary, September 29-October 1 2003. Springer-Verlag.

[155] J. Struyf, J. Ramon, M. Bruynooghe, S. Verbaeten, and H. Blockeel. Compact representation of knowledge bases in inductive logic programming. *Machine Learning*, 57(3):305–333, 2004.

[156] B. Tausend. Representing biases for inductive logic programming. In *Proceedings of the 7th European Conference on Machine Learning*, pages 427–430, Catania, Italy, April 6-8 1994.

[157] S.-A.-A. Touati, J. Worms, and S. Briais. The speedup test. Technical report, Parallélisme, Réseaux, Systèmes d'information, Modélisation - PRISM - CNRS : UMR8144 - Université de Versailles-Saint Quentin en Yvelines - ALCHEMY - INRIA Saclay - Ile de France - INRIA - CNRS : UMR8623 - Université Paris Sud - Paris XI - Laboratoire de Mathématiques de Versailles - LM-Versailles - CNRS : UMR8100 - Université de Versailles-Saint Quentin en Yvelines, 2010.

[158] R. Troncon, B. Demoen, and G. Janssens. When tabling does not work. In *Proceedings of the Colloquium on Implementation of Constraint Logic Programming Systems*, pages 18–32, Seattle, Washington, USA, August 16-21 2006. Springer.

[159] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association-rule mining. *ACM SIGMOD Record*, 27(2):1–12, 1998.

[160] M. Turcotte, S. H. Muggleton, and M. J. E. Sternberg. Generating protein three-dimensional fold signatures using inductive logic programming. *Computers & Chemistry*, 26(1):57–64, 2001.

[161] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5(1):59–68, 2003.

[162] M. Wermelinger. Conceptual graphs and first-order logic. In G. Ellis, R. Levinson, W. Rich, and J. F. Sowa, editors, *ICCS*, volume 954 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 1995.

[163] K. Yoshida and H. Motoda. Foundation of evaluation. *Journal of Documentation*, 30(4):365–373, 1974.

[164] K. Yoshida and H. Motoda. Clip: concept learning from inference patterns. *Artificial Intelligence*, 75(1):63 – 92, 1995. <ce:title>AI Research in Japan</ce:title>.

[165] K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based induction as a unified learning framework. *Appl. Intell.*, 4(3):297–316, 1994.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Mutlu, Alev
Nationality: Turkish (TC), Bulgarian (BG)
Date and Place of Birth: July 11, 1979, Shumen, Bulgaria
Marital Status: Single
Phone: 0505 237 6751
Fax: 0312 210 5544

EDUCATION

Degree	Institution	Year of Graduation
Ph.D.	Department of Computer Engineering, METU	2013
B.S.	Department of Computer Engineering, Kocaeli University	2002
High School	Kabataş High School for Boys	1998

PUBLICATIONS

- Peer-reviewed Articles in SCI Journals

  1. A. Mutlu, P. Senkul, Improving Hit Ratio of ILP-based Concept Discovery System with Memoization, The Computer Journal, 2012 (DOI:10.1093/comjnl/bxs163)
  2. A. Mutlu, P. Senkul, Y. Kavurucu, Improving the Scalability of ILP-based Multi-Relational Concept Discovery System through Parallelization, Knowledge-Based Systems, Volume 27, 2012

- Peer-reviewed Articles in International Conference Proceedings

  1. A. Mutlu, P. Senkul, Improving Hash Table Hit Ratio of an ILP-Based Concept Discovery System with Memoization Capabilities, ISCIS 2012, Paris, France, October 3-5, 2012
  2. A. Mutlu, P. Senkul, Y. Kavurucu, MPI-based Parallelization for ILP-based Multi-Relational Concept Discovery, ICMLA 2011, Hawaii, USA, December 18-21, 2011
  3. A. Mutlu, M. A. Berk, P. Senkul, Improving the Time Efficiency of ILP-based Multi-Relational Concept Discovery with Dynamic Programming Approach, ISCIS 2010, London, UK, September 22-24, 2010

- Peer-reviewed Articles in National Conference Proceedings

1. A. Mutlu, P. Senkul., Y. Kavurucu, Tumevaran Mantik Programlama Tabanli Kavram Kesif Sistemleri icin Paralel bir Yontem, 3. Ulusal Yuksek Basarimli Hesaplama Konferansi, Ankara, Turkey, April 12-14, 2012