NEW HEURISTICS FOR PERFORMANCE IMPROVEMENT OF ILP-BASED CONCEPT DISCOVERY SYSTEMS

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

BY

ABDULLAH DOĞAN

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

AUGUST 2015

Approval of the thesis:

NEW HEURISTICS FOR PERFORMANCE IMPROVEMENT OF ILP-BASED CONCEPT DISCOVERY SYSTEMS

submitted by **ABDULLAH DOĞAN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
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** _____

Assist. Prof. Dr. Alev Mutlu
Co-supervisor, **Computer Engineering Dept., Kocaeli Uni.** _____

Examining Committee Members:

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

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

Assist. Prof. Dr. Alev Mutlu
Computer Engineering Dept., Kocaeli Uni. _____

Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Department, METU _____

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

Date: _____

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

Name, Last Name: ABDULLAH DOĞAN

Signature :

ABSTRACT

NEW HEURISTICS FOR PERFORMANCE IMPROVEMENT OF ILP-BASED CONCEPT DISCOVERY SYSTEMS

Doğan, Abdullah

M.S., Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Pınar Karagöz

Co-Supervisor : Assist. Prof. Dr. Alev Mutlu

August 2015, 57 pages

A large amount of the valuable data in daily life is stored in relational databases. The accumulation of so much information motivates the need for extracting valuable patterns in relational databases. Background knowledge and a set of target examples that are stored in multiple tables are used to produce hypothesis for ILP-based concept discovery systems. Multiple arguments on these multiple tables end up large search spaces while building the hypothesis that arise computational efficiency problems.

In this thesis we focus on concept discovery systems that use Apriori-based specialization operator and work directly on relational tables. Time efficiency of these ILP systems is directly proportional to the number of queries running on DBMS. These queries mostly involve support and confidence calculation queries of candidate concept rules generated on the search space. We aim to increase time efficiency by reducing the number of running queries on these systems.

Particularly, we worked on Concept Rule Induction System (CRIS), which uses Apriori-based specialization in hypothesis construction. The methods we propose generate the same solutions as in CRIS. Therefore, we improve the efficiency without affecting the accuracy negatively.

In the first method, we prune the concept descriptors using support coverage sets. These sets are stored for memoization support of CRIS. We use the existing sets in

our proposed method so that they are also used for pruning the search space. In the second pruning method, we build cosine similarity matrix of attributes of each predicate in pre-processing step. During the specialization of concept descriptors, we prune the search space by utilizing this similarity matrix. Finally we examine the applicability of using NoSQL system MongoDB and a NewSQL system VoltDB as a storage for ILP system CRIS.

Keywords: Inductive Logic Programming, Concept Discovery, Cosine Similarity, Support, Confidence

ÖZ

TÜMEVARAN MANTIK PROGRAMLAMA TABANLI SİSTEMLER İÇİN ZAMAN PERFORMANSINI İYİLEŞTİRME AMAÇLI YENİ SEZGİSEL YÖNTEMLER

Doğan, Abdullah
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Pınar Karagöz
Ortak Tez Yöneticisi : Yrd. Doç. Dr. Alev Mutlu

Ağustos 2015 , 57 sayfa

Günlük hayatta çok miktarda değerli bilgi ilişkisel veri tabanlarında tutulmaktadır. Çok fazla bilginin toplanması, ilişkisel veri tabanlarında değerli modellerin bulunmasını teşvik etmektedir. Tümevaran mantık programlama tabanlı keşif sistemlerinde, çoklu tablolarda tutulan arkaplan bilgisi ve hedef örnekleri kullanılarak hipotezler üretilir. Bu çoklu tablolardaki çoklu argümanlar hipotez üretirken geniş arama alanlarına dolayısıyla hesaplama verimliliği problemlerine sebep olurlar.

Bu tezde, Apriori tabanlı özelleştirme operatörü kullanan ve ilişkisel tablolar üzerinde çalışan keşif sistemleri üzerinde yoğunlaştık. Bu sistemlerdeki zaman verimliliği veritabanı yönetim sisteminde çalışan sorgu sayısı ile doğru orantılıdır. Bu sorgular genellikle arama alanında oluşan aday kavram kurallarının kapsam ve doğruluk hesaplama sorgularıdır. Bu sistemdeki çalışan sorguları azaltarak zaman verimliliğini arttırmayı amaçlıyoruz.

Özellikle hipotez üretilmesi sırasında Apriori-tabanlı özelleştirme kullanan Kavram Kural Tümevarım Sistemi (CRIS) üzerinde çalıştık. Geliştirdiğimiz yöntemler CRIS ile aynı sonuçları üretmektedir. Dolayısıyla doğruluğu olumsuz etkilemeden verimliliği arttırmaktadır.

İlk metodda kavram tanımlayıcılarını destek ölçütü kapsama kümelerini kullanarak budadık. Bu kümeler CRIS'te tablolama desteği için kullanılmaktadır. Metodumuzda varolan bu kümeleri, arama alanında da budama gerçekleştirecek şekilde kullandık.

İkinci budama metodunda, ön işleme olarak tabloların tüm niteliklerinin kosinüs benzerliği matrisi oluşturduk. Kavram tanımlayıcılarının özelleştirmesi aşamasında kosinüs matrisini kullanarak arama alanında budama yaptık.

Son olarak bir tümevaran mantık programramı olan CRIS'in bir NoSQL sistemi olan MongoDB ve NewSQL sistemi olan VoltDB'yi depolama alanı olarak kullanabilirliğini araştırdık.

Anahtar Kelimeler: Tümevaran Mantık Programlama, Kavram Keşfi, Kosinüs Benzerliği, Destek Ölçütü, Güven Ölçütü

To my wife and daughter

ACKNOWLEDGMENTS

I would like to thank to my supervisor Assoc. Prof. Dr. Professor Pınar Karagöz for her guidence, patience and friendship during the development of this work. I am so grateful to Assist. Prof. Dr. Alev Mutlu for being a colleague and then my co-supervisor. His contributions helped me alot. I would also like to thank to Professor Nihan Kesim Çiçekli, Professor İsmail Hakkı Toroslu and Professor Ahmet Coşar for being comitee members in my jury.

I am grateful to all of my colleagues at METU for providing a friendly atmosphere. I am especially thankful to my friends Murat Öztürk, Ahmet Rifaioğlu and Aybike for being family friends and giving motivation throughout the thesis.

I also cannot pass without mentioning the people who built this green campus environment.

Finally, I would like to thank to Fatih Semiz who worked with me late nights at A-206.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ILP	Inductive logic programming
SQL	Structured query language
RDM	Relational data mining
CRIS	Concept rule induction system
DBMS	Database management system
CRIS w-EF	Concept rule induction system with extra features

CHAPTER 1

INTRODUCTION

Inductive logic programming is a discipline that has roots in both machine learning and logic programming. With the use of theoretical bases of logic, it overcomes the limitations of propositional learners and find a relational representation for background knowledge [47].

Different approaches are researched in this area based on the search direction as either top-down or bottom-up. The former starts with most general clauses and applies specialization operators to achieve the goal. The latter starts with most specific clauses and applies generalization techniques to achieve the goal. Both methodologies use logic-based operators.

These systems find place in wide range of fields. Since from their first definition, they have been applied to engineering [17, 22], biochemistry [35, 44, 36], language processing [83, 12]. Also several performance improvement techniques are searched in these area. All aim to find the goal without loss valuable information in less time. Such improvement techniques including reordering the literals in a clause [78], applying parallelization techniques [26], and caching techniques [12] or using query transformations [10, 11].

In this thesis, two pruning methods are presented. Both are developed under Tabular CRIS w-EF [55, 53], an ILP algorithm that uses Apriori-based specialization operator. The aim in both pruning methods is to reduce the number of SQL calls to the database because of the generated large search space due to the refinement operator.

In the first method, memoization structure of the existing algorithm extended. Cov-

erage set of support values of parent concept descriptors which are already stored in hash table are used to prune the specialized concept descriptors. The motivation can be explained in terms of relational algebra. Support of a concept descriptor of length l is a relation instance that is the result of algebraic queries which consist of selections (σ) and joins ($\bowtie$) applied to database relations and projected (Π) based on head arguments of the concept descriptor. Since a "$l + 1$" length clause is generated from two length "l" clauses, support of the $(l + 1)$ length clause can be generated using the support instances of length l clauses. It is the result of joining length l queries thus intersecting them.

The second pruning method is inspired from information retrieval. Cosine similarity of argument domain vectors are used to prune concept descriptors without calculating support queries.

In this thesis, we also present the applicability of the existing ILP system on a NoSQL database and NewSQL database. We explore the feasibility of using an embedded data model in MongoDB. In addition, we examine the performance of running asynchronous query calls to multiple clusters in VoltDB. Lastly, we perform tests by changing MySQL tables to $Memory$ storage engine.

Chapters of this thesis are organized as follows. In Chapter 1, we briefly define the problem with emphasis on our motivation behind this thesis. Chapter 2 provides related work on ILP-based systems. In Chapter 3 we provide background about inductive logic programming. Also we focus on the algorithm defined for CRIS and related work on CRIS. In Chapter 4 we present our two pruning methods which are extensions to CRIS wEF. Also in this chapter, we examine applicability of CRIS in different database management systems like MongoDB and VoltDB. In Chapter 5 experimental results are presented, and in Chapter 6 the thesis is concluded with an overview and final remarks.

CHAPTER 2

LITERATURE SURVEY

In this chapter first we introduce related work on ILP-based concept discovery systems. Then performance improvement techniques applied to ILP-based systems are presented.

2.1 Concept Rule Induction Systems

2.1.1 FOIL (First Order Inductive Logic Learner)

FOIL [59, 60] is a variant of sequential covering algorithm. It starts with a literal with just left hand side, then adds a literal one at a time (top-down). When it finds a rule, it removes positive examples which are covered by this rule then advances to learn another rule. It supports recursive rules where the target relation can be found in the body of the clause.

While adding a literal to the right hand side, FOIL follows the rules below:

- New literal should have at least one bounded variable.
- If it is a recursive rule (predicate of the new literal is same as the literal on the left hand side) then restrict possible arguments to dissallow some problems related to recursion.

To decrease the coverage of negative examples and increase coverage of positive examples, FOIL uses gain metric for evaluating literals.

2.1.2 WARMR

An apriori based ILP system WARMR is defined in [3]. It is an extension of APRIORI algorithm and is modified to be used for multiple relations [14, 15, 38]. It is the earliest ILP algorithm that is applied to chemical components data [37] where patterns for carcinogenicity relations of chemical compounds are discovered. In addition, it is applied to telecomuncation network analysis and part-of-speech tagging of natural language text [14]. It uses θ-subsumption based generalization [16], starts with the most general patterns and iterate through generalization to evaluation phases where frequencies are calculated. Language bias, minfreq. (minimum support threshold) and a set of examples (E) are given as input and patterns in language bias that cover above the minfreq. of examples (E) are discovered.

2.1.3 GOLEM

GOLEM [48] is a relational ILP based concept discovery system which uses relative least general generalization (rlgg) to guide the search space. GOLEM contains two nested loops. In the outer loop, the clauses which cover the positive examples are randomly picked. Following in the inner loop, RLGG of uncovered positive examples are computed and the one with the highest coverage is selected. GOLEM is applied in various applications such as satellite fault diagnosis model [22], mesh design [17], qualitative physics model design [7].

2.1.4 PROGOL

PROGOL [45] is a top-down ILP algorithm. It uses inverse resolution adapted [46] to first order logic for generating hypothesis. It uses sequential covering algorithm as in FOIL, but also uses mode declarations that define restrictions on predicates.

2.1.5 CRIS

CRIS is a top-down ILP based concept discovery system where target is a single table and background facts are multiple tables that reside on a database system [33, 49]. It first generates most general hypothesis and then specializes the concept descriptors using apriori based specialization operator. It uses support and confidence for pruning infrequent rules. It finds the best concept descriptor using f-metric [28]. As in sequential covering algoritm, at each iteration CRIS removes target instances that are covered by the best concept descriptor. Since efficiency improvements for CRIS are proposed in this thesis, detailed information is given in Chapter 3.

2.2 Time Performance Improvement Techniques

ILP-based concept discovery systems suffer efficiency problems because of the generated large search spaces. Therefore performance improvements has received a lot of interest on these systems. In this section efficiency improvement methods applied to ILP-based systems are introduced.

2.2.1 Query Packs

Blokeel et al. improves the efficiency of ILP using query packs [6]. In the proposed method, literals with identical prefixes in the search lattice are grouped together to form query packs. It is a tree structure where a query pack is computed once and it's results are used by their successors. As an example five queries given below:

```
p(X), I = 1.
p(X), q(X,a), I = 2.
p(X), q(X,b), I = 3.
p(X), q(X,Y), t(X), I = 4.
p(X), q(X,Y), t(X), r(Y,1), I = 5.
```

It is converted to disjunctive query as follows:

p(X), (I=1 or q(X,a), I=2 or q(X,b), I=3 or q(X,Y), t(X), (I=4 or r(Y,1), I=5))

Since the literal P(X) is the prefix for all the queries, it is evaluated only once and performance improvements can be achieved according to the cost of evaluating P(X). TILDA and WARMR systems are re-implemented for query packs and experiments are conducted on mutagenesis data set [77]. The results show remarkable speedups achieved by applying query packs.

2.2.2 Query Transformations

Costa et al. proposed four query transformations in [10, 11]. In their work, first order predicates are transformed into equivalent predicates in an efficient form. In Theta-transformation (t_θ) redundant literals in the body are eliminated based on subsumption relation. In Cut-transformation ($t_!$) dependent literals in the body of the clause are partitioned to form equivalent classes. Each equivalent class is computed independently. If there is no solution in a class, then there is no solution for all classes in the clause. The third transformation called the once-transformation (t_o) has same effect as cut-transformation, but partitioning process is improved by using prior knowledge. The fourth transformation, smartcall-transformation (t_s) uses the fact that coverage list of refined clauses are subset of coverage list of their parents.

2.2.3 Reordering Literals

In [78], optimization of ILP system ALEPH is done by reordering literals. The proposed method first estimates average execution time costs of the literals. Then moves the literals with lowest estimate first in the clause. Experiments on carcinogenesis data set [76] verifies the minimum average execution time of the clauses after reordering.

2.2.4 Caching

In [12] performance improvement is achieved using memoization in P-Progol. Positive and negative coverage sets of clauses are stored in cache to be used in subsequent search of the similar clause. In addition, a cache structure for pruned clauses are

stored in "prune cache". The computational gain here is that; if a new clause is in the cache then prune it without any calculation. Caching improved the time efficiency of the ILP system. In addition to caching, this system uses constraints to limit the search space. One of the constraints defined is related to the length of the clause; length should not be greater than 5. Another constraint is defined for pruning the search space; a clause should cover at least 15 positive target examples. Experimental results show that caching brings 15.75 speed-up and for small data sets 10.74 speed-up. Additional experiments [25, 24] for for coverage caching are conducted by using the another prolog based ILP system April [23]. Results show excessive memory usage with the use of caching.

Tabular CRIS is another ILP system that uses cache [50]. In this system, support and confidence query results are stored in hash tables. Repeating calls of an SQL query is retrieved from the hash. More details about this system is given in Chapter 3 Background.

2.2.5 Parallelization

Fonseca et al. proposed a pipelined data-parallel algorithm $P^2 - mdie$ developed on ILP system April [23] to be used in distributed memory machines [26]. In the proposed system target examples are split to workers. In addition, learning step is split to stages and each stage uses only examples reserved to them. Finding the best rule is done by pipelining. After a good rule is found in a stage , it is sent to another stage that uses different part of the examples. At the end of the pipelining rules found are send to the master.

There are several researches about parallelization of existing ILP systems. Parallel extension for ILP system CLAUDIEN is described in [13]. The task is Parallelization of C4.5 system is described in [39].

2.2.6 Language Bias

In [57] efficiency improvement of ILP systems is achieved by using restrictions to limit the generated hyphotheses namely declarative bias specifications.

CHAPTER 3

BACKGROUND

In this chapter basic topics about this thesis is given. Since many relational data mining algorithms have their roots in inductive logic programming, we first give definition of relational data mining. Then detailed structure of CRIS is given. Lastly we present information NoSQL and NewSQL systems database systems.

3.1 Relational Data Mining

Multi-relational data mining models techniques for querying, manipulating or storing complex information in relational database [38, 20]. Tables and columns are main components of these systems. The propositional algorithms look for patterns in a single table. On the other hand, searching for valuable information in multiple tables is the subject of multi-relational data mining. It has roots in inductive logic programming which provides expressive language. It has been applied to wide range of areas where data is stored in relational tables, bioinformatics, web mining, finding patterns in business etc.

3.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a field in machine learning which aims to discover patterns from given examples and knowledge from experience [47]. It comes from two disciplines. Finding hypothesis in inductive manner is the area of inductive machine learning. The representation of discovered patterns and background knowl-

edge in ILP are represented by computational logic, a subset of first order logic [20].

Given background knowledge (B), positive (P) and negative (N) examples (E), hypothesis (H) in an ILP system should be complete B $\wedge$ H $\models$ P and consistent B $\wedge$ H $\not\models$ N [20].

An ILP problem on sample data set *elti* is given in Table 3.1 where *elti* predicate instances are the target and background knowledge is *wife* and *brother* predicate instances.

Table 3.1: Sample ILP problem

Examples	Background Knowledge	
elti(nalan,bedriye) $\oplus$	wife(ayten,ismail)	brother(yildirim,sadullah)
elti(cemile,ayten) $\oplus$	wife(nalan,sadullah)	brother(mehmet,ismail)
	wife(bedriye,yildirim)	
	wife(cemile,mehmet)	

These systems commonly use sequential covering algorithm. In this algorithm hypothesis are generated iteratively where coverage set of a hypothesis is removed from positive examples. Then the algorithm advances building the next hypothesis until all positive examples are removed.

A Generic ILP Algorithm 1 defined by Muggleton et al [47], candidate hypothesis (QH) are stored in a queue. At each iteration, a hyphotesis (H) is deleted from queue and inference rules are applied to the deleted one to generate new hypothesis. Then newly generated hypothesis are added to the queue. Only promising hypothesis continue to exist in the queue for the next iterations by applying a pruning method. Iteration repeats until it meets the specified stop criteria.

Search space for ILP algorithms is based on θ-subsumption [21]. If a substitution θ is applied to clause c and c$\theta \subseteq$ c$^{'}$ then c θ-subsumes c$^{'}$. In addition, if θ-subsumes c$^{'}$ then c entails $c^{'}$ (c $\models$ c$^{'}$), also c is at least as general as $c^{'}$ ($c \leq c^{'}$). If $c < c^{'}$ then c is more general then of $c^{'}$ therefore $c^{'}$ a refinement of c. Consider two clauses P_1 and P_2 and their refinement C_1.

P_1:elti(A,B):-brother(C,D)
P_2:elti(A,B):-daughter(C,A)

QH := Initialize

repeat

Delete H from QH

Choose inference rules $r_1, r_2...r_k \in$R to be applied to H

Apply rules $r_1, r_2...r_k$ to H to yield $H_1, H_2...H_n$

Add $H_1, H_2...H_n$ to QH

Prune QH

until Stop criteria (QH) satisfied

Algorithm 1: Generic ILP algorithm [47]

C_1:elti(A,B):-brother(C,D), daughter(E,A)

Clause P_1 θ-subsumes the clause C_1 and since when applying θ as empty substitution to P_1, it is a subset of C_1. Also $P_1 < C_1$ so P_1 is a generalization of C_1.

Generating hypothesis is done either by top-down manner or bottom-up manner. In top down systems, refinement graph search operation is done by θ-subsumption. Firstly, most general clauses are generated then they are specialized in a way that they cover positive examples and no negative examples. In bottom-up systems, examples and background knowledge is used to create least general generalizations based on θ-subsumption. Firstly most specific clauses are generated then they are generalized in a way that they cover positive examples and no negative examples.

3.3 Concept Rule Induction System (CRIS)

Two proposed pruning methods in this thesis are embedded into Tabular CRIS-wEF, which is an extension of CRIS algorithm. In this section we give more information about CRIS. Firstly, we present the main stages of the algorithm. Then developments on this algorithm is presented.

3.3.1 Algorithm of CRIS

CRIS is a top-down ILP algorithm. It starts with the generalization phase. Then iteratively follow specialization, filtering and covering phases.

Generalization In this step, CRIS generates most general hypothesis. It first determines available constants and variables in arguments of target and each background predicates. Then target and background predicates with their feasible arguments are combined to form a one head one body concept descriptor. Minimum support threshold value multiplied by count of rows in a predicate is determinative for choosing if a nominal attribute in a predicate can be constant or not.

As an example, consider a predicate $pred1(nomArg1, numArg1)$ where *nomArg1* is a nominal argument and $numArg1$ is a numeric argument. If *minimum support threshold* is given as 0.3 and count of rows is 5894, then possible constant values for nominal attributes calculated as in Table 3.2

Table 3.2: Calculating possible values for a nominal argument

SQL	SELECT nomArg1 FROM pred1 GROUP BY nomArg1 HAVING COUNT(*) >=0.3 *5894;
Sample output	1 2
Feasable values	1 2 VARIABLE

For numeric attributes, instead of searching feasible constant values, possible range values are calculated using number of rows in the predicate and number of target instances. Supposing that 4595.th row is calculated as a partitioning point for the predicate $pred1$. Possible starting range values for numeric attributes can be calculated as in Table 3.3

Note that, this range values are combined with ">=" and "<=" operators in the predicate.

Table 3.3: Calculating possible values for a numeric argument

SQL	SELECT * FROM (SELECT numArg1 FROM pred1 ORDER BY numArg1 limit 4595) a ORDER BY numArg1 DESC LIMIT 1;
Sample output	0.047
Feasable values	>=0.047 <=0.047 VARIABLE

Combining all feasible values for arguments of a predicate results in most possible form of it. Sample most general form of the predicate *pred1* is shown in Table 3.4 without renaming of the variables.

Table 3.4: Generalization of $pred1(nomArg1, numArg1)$

Pred1 (nomArg1,numArg1)
Pred1 (1, >=0.047)
Pred1 (1, <=0.047)
Pred1 (1, VARIABLE)
Pred1 (2, >=0.047)
Pred1 (2, <=0.047)
Pred1 (2, VARIABLE)
Pred1 (VARIABLE, >=0.047)
Pred1 (VARIABLE, <=0.047)
Pred1 (VARIABLE, VARIABLE)

Specialization In this step each concept rules generated at previous step are unified using θ-subsumption. Candidate generation is based on Apriori-based specialization operator. If a concept descriptor has one different body literal from any other concept descriptors, then these concept descriptors are unified to form a new concept descriptor with one more body literal.

$$C_1 \cup C_2 = \{C_1 \cup I_{21} | C_{12} = C \cap C_2\theta - C_{21} = I_{21}\} \tag{3.1}$$

Filtering Quality measure to choose interesting rules from all possible rules in CRIS

is based on their support and confidence values. According to the these values, a rule is either pruned, added to the solution set or further refined.

Definition 1. *Support of a rule is the number of positive target examples explained by the rule divided by number of target examples [56].*

$$Support(h \leftarrow b) = \frac{\mid \text{bindings of variables for h that satisfy h} \leftarrow \text{b} \mid}{\mid \text{bindings of variables for h that satisfy h} \mid} \quad (3.2)$$

Definition 2. *Confidence of a rule is the number of positive target instances explained by the rule divided by number of instances that are deducible from the rule [56].*

$$Confidence(h \leftarrow b) = \frac{\mid \text{bindings of variables for h that satisfy h} \leftarrow \text{b} \mid}{\mid \text{bindings of variables for h that satisfy b} \mid} \quad (3.3)$$

In order to select considerable rules from the set of all possible rules the CRIS uses the constraints listed below:

- A possible rule that has support and confidence values higher then minimum thresholds is added to the solution set.
- Prune the possible rules that have support less then the threshold.
- If support value of the possible rule is less then minimum threshold but confidence value is higher then two of its parents then it is further refined.
- A possible rule that has confidence value less then any of its two parents, then it is pruned.

Covering The best rule in the solution set is selected using f-metric. As in sequential covering algorithm, target tuples that are satisfied by the best rule are removed in this step.

Iteration continues until all target instances are covered by produced solutions, no concept descriptors are produced or maximum depth is reached.

3.3.2 Enhancements of CRIS

There are several studies for improving efficiency of CRIS. One of the study is given in [50] where a dynamic programming based approach of CRIS named Tabular CRIS is proposed. The aim is to prevent repeating queries in the DBMS, since different concept descriptors may have the same query correspondence. Tabular CRIS uses two hash structures (for support and confidence) in the form <query,int> that stores results of support and confidence queries. This queries are aggregate queries and all begin with "SELECT COUNT". Before requesting the result of the support/confidence query from DBMS, Tabular CRIS first checks whether result of the query is already the hash. If it finds the result in the hash then uses this result in the hash. Otherwise sends query to the DBMS and stores the result in the for later use. The proposed system is applied to different types of data sets and achieved high rates of hash table hit ratios from 14% to 91%.

In [55, 53] Tabular CRIS-wEF is proposed. Support and confidence queries of concept descriptors are changed to allow returning the rows instead of counts. The queries changed from "SELECT COUNT" to "SELECT DISTINCT" form and hash structures also changed to store the values, in the form <query, resultset>. Also, covering algorithm is modified to remove covered tuples from resultset parth of the hash tuples. By these modifications, hash table hit ratio increased by catching repeating queries at different epochs. Experiments show that, hash table hit ratio of datasets that run in more than one epochs increased by this approach [53].

Mutlu et al. in [54, 56] proposed parallel version of CRIS namely pCRIS. In the proposed method, splitting the job is done by exacly one master process and works are done by multiple worker processes. Parallelization is applied to two time consuming steps of CRIS. Firstly in specialization step where clauses of length l are unified to form length $l+1$. Secondly in filtering step where support and confidence queries are send to tb DBMS and evaluation of the clauses are done. The proposed algorithm is applied to seven data sets and shown that algorithm performs better than several ILP-based parallel learning systems for large data sets.

In [52] a hybrid graph-based concept discovery is proposed. It is a hybrid system

that utilizes both substructure based and path-finding base aproaches. It uses acyclic directed graph where nodes are arguments of the predicates. Experiments conducted on four kindship datasets to compare the results with CRIS. Both generate same solutions on two of the datasets and semantically identical solutions on one of them. Another experiment conducted on family data set to compare the results with Relational Paths Based Learning (RPBL) [27]. The proposed method generates more concept descriptors than RPBL and generates concept descriptors with lesser clause length.

3.4 Vector Space Model and Cosine Similarity

Information retrieval (IR) is a subfield of computer science which is interested in representation, storage and access of unstructured information by searching within relational databases, documents, text, multimedia files, and the World Wide Web [19, 79]. The main purpose of information retrieval model is to “finding relevant knowledge-base information or a document that fulfill user needs" [63].

An important and widely used IR model is vector-space model. In vector-space model, the documents are represented as vectors in a common vector space [41]. Vector Space Model uses term frequency and inverse document frequency which is known as tf-idf weighting. In tf-idf, term frequency (tf) is the number of times that the term occurs in document or query texts and an inverse document frequency (idf) is the inverse of the number of documents that contain the term [32].

In vector-space model, documents are ranked by some similarity value based on the user query and the documents. [18, 61, 64, 5]. In this model, angle between the vector representation of documents and user query is calculated by cosine function.

The cosine similarity is generally used to compute the similarity between these two vectors. The formula of cosine similarity is given in equation 3.4 as follows:

$$sim(dj, q) = \frac{dj.q}{\mid dj \mid\mid q \mid} = \frac{(\sum_{i=1}^{N} w_{i,j} w_{i,q})}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{3.4}$$

$$d_j = (w_{i,j}, w_{2,j}, ...w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, ...w_{t,q})$$

3.5 NoSQL

Software industry uses relational databases their data for a long time. RDBMSs prove themselves by supporting concurrency control, transactions, standard query language, rich interfaces for reporting and integration mechanism [62]. However these systems have impedance mismatch problems such that data stored relational model should be translated into memory structures of the programs. In addition, these systems have efficiency problems while storing large volumes of data on clusters.

With the rise of Web 2.0 and cloud technology, performance and scalability in data stores needs brings new technology, namely Not Only SQL or NoSQL in short. Most of these systems are open source and are designed to run on clusters. They are not relational and have no schema, adding a field to a record does not need changing the structure.

They are categorized into four categories. The first one is Key-Value databases which store blob values that are uniquely identified by a key. User can either, add, delete or retrieve the value of the key. There is no type restriction on the value, applications are responsible for parsing it. The main advantage of these systems is scalability and efficiency, the data can easily be distributed by adding more servers to the system. Scalaris [73], Amazon DynamoDB [4], Voldemort [75], Redis [70] and MemcacheDB [71] are some of them.

Secondly, document databases are subclass of the first category, which differs by the structure of the values. In these systems, values are documents in the form XML, JSON or BSON etc. that changes based on the database system. The data stored

inside the document should obey the rules of the form structure that the database chooses. Since these structures enable storing flexible values, one can store scalar, the other can store collections or any hierarchical tree value. MongoDB [72], OrientDB [80], CouchDB [66], Couchbase [68] are some of the systems in this category.

Figure 3.1: Column family databases

Thirdly, in column family databases, data is stored by columns instead of rows. Because of storing values of columns together (see Figure 3.1), aggregations on columns are much faster than relational tables. Also, because they have the same type, compression can be applied to the columns. Cassandra [65], HBase [67], BigTable [31] are some of the systems in this category.

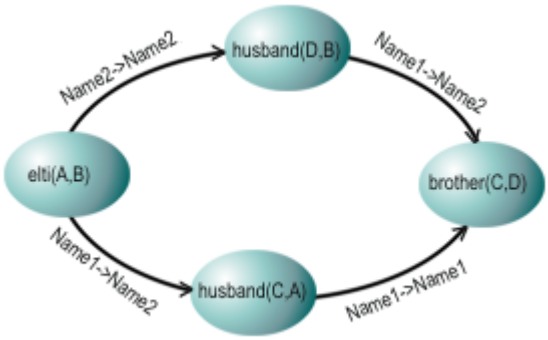

Figure 3.2: Graph databases employ nodes, edges etc.

The last category, graph databases enable to store elements (nodes) and their relations (edges) (see Figure 3.2). Traversing is done by graph-like queries. Adding more than one edges is possible, there is no restriction on type and number of edges defined. Neo4J [81], AllegroGraph [30] and FlockDB [74] are some of the systems in this category.

Table 3.5: Common MongoDB data types

String,	Double,	Object,
Array,	Binary data,	Object id,
Boolean,	Date,	Null,
32-bit integer,	64-bit integer,	Timestamp,
...		

3.6 MongoDB

MongoDB is one of the common known document type open-source NoSQL database [43]. Data is stored in the form of BSON documents (a JSON-like format). Documents have field-value pairs (see Figure 3.3) where value field can be array, document, array of document or any BSON type. Documents are stored in collections. For structures larger than 16MB, MongoDB supports GridFS structure. It splits the file into chunks and stores them separately.

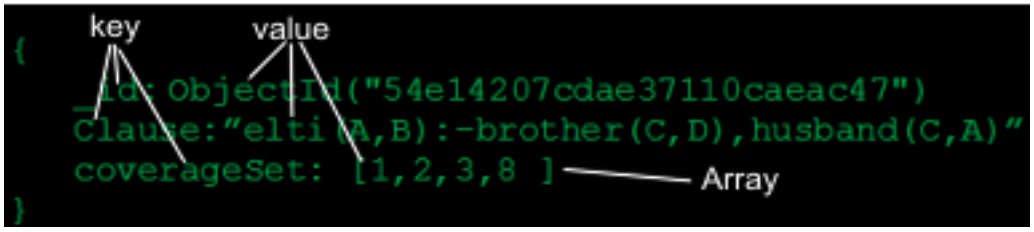

Figure 3.3: Key-value pairs

Unlike relational databases, a collection is created at the first insert operation. Thus, there is no need to define a data type for fields. Every document has $\_id$ field as primary key. If not explicitly given, system sets a unique value with ObjectId type. Common types used in MongoDB is given in Table 3.5 and relational terminology counterparts are given in Table 3.6.

There are two types of document structures used for representing relationships between documents, "references" and "embedded data" (see Figure 3.4). In the former, two documents are connected by adding a link field from one document that references target primary key. In the latter, target document is embedded into a field or array. Embedded model is generally used in one-to-one and one-to-many relationships. Retrieving or updating child document is single atomic operation, since it is

Table 3.6: Relational terminology vs. MongoDB terminology [43]

Relational DBMS Terms	**MongoDB Terms**
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	embedded documents and linking
primary key:any unique column or column combination as primarykey.	**primary key**: _id field is automatically set as the PK
aggregation	aggregation pipeline

inside the parent document but growth of the data may cause problems. Using embedded model in many-to-many relationships causes duplication. Therefore references model is used in many-to-many type relationships.

Because of the flexible schema, tree structures can also be stored MongoDB database:

- using parent reference inside child
- using references of child nodes as array inside parent
- using references of all ancestors inside child

The database supports a rich query language for retrieving and modifying data stored in BSON format. For querying, it supports ordering, limiting rows, allows projecting only necessary fields, adding conditions, using aggregate functions etc.

3.7 NewSQL

NewSQL systems are combination of relational DBMSs and NoSQL systems. They support relational model and SQL of RDBMSs, horizontal scaling of NoSQL systems [29]. VoltDB [82], Google Spanner [9], Clustrix [8] and NuoDB [2] are some of NewSQL systems. Although they serve tables of relational systems, the underlying representation of data may differ from each other.

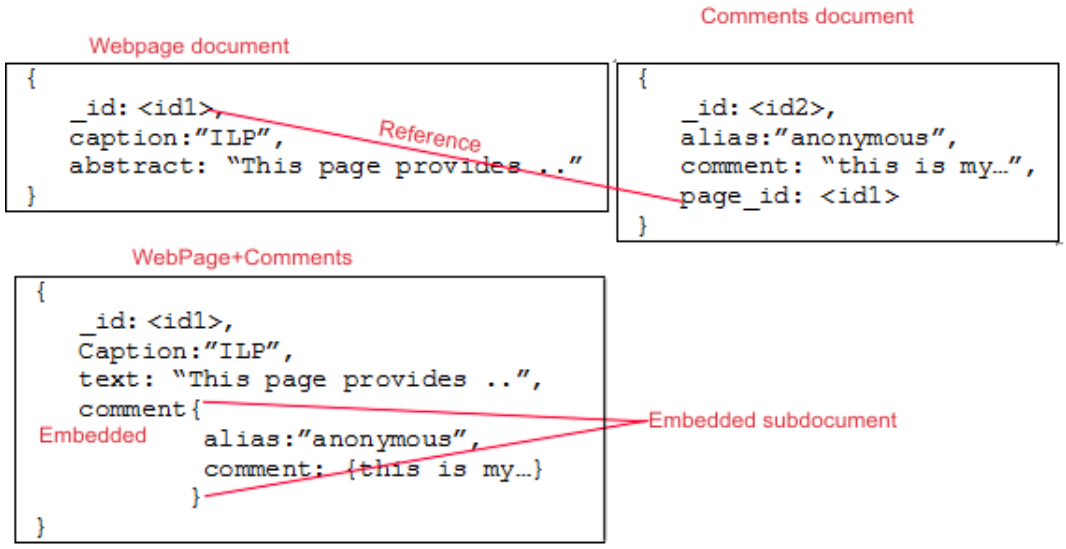

Figure 3.4: Reference and embedded type relationship

The support for SQL queries also differs according to the vendors. Some have restrictions on standard queries such as in aggregation; e.g. VoltDB does not support using "having" clause. Also in previous versions of VolTDB it was obligatory to use stored procedures while client interface interacts with the database or creating a table could not be done at the console etc. Since it is a new technology developments on these systems are still in process, every new version comes with lesser restrictions.

3.8 VoltDB

VoltDB is an in-memory NewSQL database system [82]. It supports ANSI standard SQL. It is available in paid enterprise edition and a free open source edition with some missing features. VoltDB supports snapshots to save the data to disk for later use. Since it is an in memory database, once it is shutdown, the data removed from the memory.

Figure 3.5: Partitioning

It allows partitioning, which splits rows of a table to several nodes. A table can have at most one partition with one or more columns. If there is a primary key, it must be included in the partitioning columns. VoltDB also allows replicating tables to all nodes. It is suitable for readonly and small tables, those are frequently queried. If a table is not partitioned then it is replicated by default.

VoltDB supports asynchronous procedure calls to the database. Traditional client applications send requests to the DBMS and wait for the response, they cannot continue processing until they got response. By using asynchronous calls, the client application does not have to wait for the response, can continue sending another requests. When database is ready, it notifies the client by a callback procedure and the client gets the response. Every node has its own queue for transactions, so connecting to multiple nodes with asynchronous calls allows the client to distribute the work and increase throughput.

CHAPTER 4

PROPOSED METHODS

The major problem of using ILP-based concept discovery systems is that, the use of Apriori-based approach generates rules excessively. For a data set that has predicates with so many arguments, there may be thousands of possible rules. Each rule has query costs in a DBMS while calculating support and/or confidence metrics. Performance problems reveal the need for efficient pruning methods applied to minimize wasted effort on DBMS, without decreasing the accuracy.

In this chapter, we describe two different heuristic for pruning the queries for rule quality assessment. In addition, we present our efforts on adapting NoSQL and NewSQL technologies for ILP-based concept discovery.

4.1 Utilizing Coverage Lists as a Pruning Mechanism for Concept Discovery

In this thesis we focus on computational efficiency problems that arise due to the large search spaces of ILP-based concept discovery systems. We propose a pruning mechanism to reduce the size of the search space. In the proposed method [51], using coverage sets as a memoization technique is extended to be able to be used as a pruning mechanism also. The method makes an assumption: if two hypotheses are refined via an Apriori-like operator, the coverage set for support of a refined concept descriptors should at maximum be the intersection of the coverage sets of its parents.

The proposed algorithm is embodied into Tabular CRIS-wEF [55, 53]. It is an ILP based concept discovery system that has memoization capabilities and uses Apriori-

like refinement operator and uses support and confidence as quality measures. The experimental results show that the proposed method decreases the search space 4-22%. The results are promising for the reason that pruning is done before the calculation of support values of concept descriptors from the DBMS.

4.1.1 Motivation

The motivation behind the proposed algorithm is that if an ILP system uses Apriori-like refinement operator and possible rules are pruned according to their support values, coverage set cardinality of a possible rule that counts for its support can not be more than cardinality of its two parents' coverage set intersection.

A support query of length k clause "$T : -B_1, B_2, ..., B_k$" can be represented as
$\pi_{projection\_columns}(S_1 \bowtie B_k)$
where B_k is a predicate in its body and projection_columns are the arguments of target predicate T. S_1 is defined as:
$S_1 = \sigma_{selection\_conditions}(T \bowtie B_1 \bowtie B_2 \bowtie ..B_{k-1})$
where selection_conditions are the constants applied to the arguments of the predicates, T is the target predicate and $B_1, B_2, ..B_{k-1}$ are the background predicates except from B_k.

Two length k clauses L_1 and L_2 that have only one different literal in the body are unified to form clause with length $k + 1$. Its support query can be presented as:
$C_1 = \pi_{projection\_columns}(S_1 \bowtie B_{1k} \bowtie B_{2k})$
where S_1 presents same literals of L_1 and L_2. B_{1k} and B_{2k} two the different literals. Since support set of L_1 is $\pi_{projection\_columns}(S_1 \bowtie B_{1k})$ and suppport set of L_2 is $\pi_{projection\_columns}(S_1 \bowtie B_{2k})$, if B_{1k} and B_{2k} does not have same variables in their arguments, C_1 should be the intersection of the support sets of L_1 and L_2. In the case where they have same variables, C_1 should be a subset of intersections of support sets of L_1 and L_2.

Consider *elti* data set that has predicates as given in Table 4.1. Two clauses generated on this data set, named P_1 and P_2 are given in Table 4.2 where first arguments of the predicates are $name1$ and the seconds arguments are $name2$. Using Definition 1,

Table 4.1: Predicates in Elti data set

Predicate name	**Argument names**
elti	name1,name2
brother	name1,name2
husband	name1,name2
mother	name1,name2
sister	name1,name2
son	name1,name2
wife	name1,name2
daughter	name1,name2
father	name1,name2

Table 4.2: Two Possible Clauses at Depth 2

Two Possible Clauses with length 3	
P_1	$elti(A,B) : -brother(C,D), husband(C,A)$
P_2	$elti(A,B) : -husband(C,A), husband(D,B)$
Refinements of the clauses	
C_1	$elti(A,B) : -brother(C,D), husband(C,A), husband(E,B)$
C_2	$elti(A,B) : -brother(C,D), husband(C,A), husband(C,B)$
C_3	$elti(A,B) : -brother(C,D), husband(C,A), husband(D,B)$

query for the coverage set of P_1 can be formed as seen at Figure 4.1 where $projection$ of the query comes from the head part of the clause and $selection$ is formed due to the similar naming of the arguments from both head and body.

At specialization step, these two clauses are unified to form clauses with length 4. The refinements (C_1, C_2, C_3) of these two parent clauses (P_1, P_2) can also be seen at Table 4.2. In these refinements, first two literals come from P_1 and the last literal come from P_2. In C_1, each literal is bounded to the same arguments of predicates just as their counterparts at P_1 and P_2.

Different from C_1, relationship of arguments of C_2 and C_3 differs from their counterparts in P_2. To incorporate all background facts, refinement operator generalizes the predicates in all possible ways. In C_2, $name1$ of 4$^{\text{th}}$ literal (husband(C,B)) is bounded to $name1$ of the 3$^{\text{rd}}$ literal ((husband(C,A)) which is not bounded to the same argu-

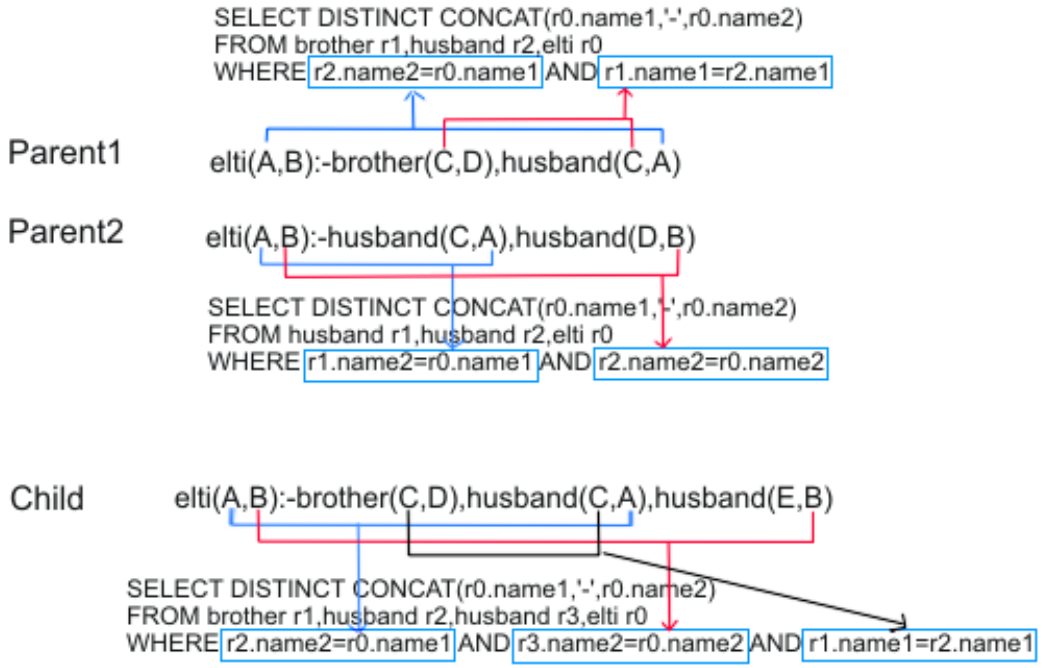

Figure 4.1: Two clauses and their refinements

ment of the same predicate in P_2. Also in C_3, $name1$ of 4th literal (husband(D,B)) is bounded to $name2$ of the 2nd literal ((brother(C,D)) which is not bounded to the same argument of the same predicate in P_2. In these cases correct sets cannot be produces by intersecting coverage sets of P_1 and P_2.

To overcome such conditions, refined concept descriptors are analyzed to explore parents with same bindings. Table 4.3 shows the extracted concept descriptors from the refined concept descriptors C_2 and C_3.

In Table 4.3, clause C_2 is pruned without querying the support value from DBMS. Intersection coverage sets of its parents P_1 and P_{C_2} is empty set.

Table 4.3: Extracted parent concept descriptors and their coverage sets

Length 4	Length 3
C_1:$e(A,B) : -b(C,D), h(C,A), h(E,B)$ intersection of coverage set (1,2,3,4,5,6,7,8)	$P_1 : e(A,B) : -b(C,D), h(C,A)$ coverage set (1,2,3,4,5,6,7,8) $P_2 : e(A,B) : -h(C,A), h(D,B)$ coverage set (1,2,3,4,5,6,7,8)
C_2:$e(A,B) : -b(C,D), h(C,A), h(C,B)$ intersection of coverage set () Pruned	P_1:$e(A,B) : -b(C,D), h(C,A)$ coverage set (1,2,3,4,5,6,7,8) $P_{C_2} : e(A,B) : -b(C,D), h(C,B)$ coverage set ()
C_3:$e(A,B) : -b(C,D), h(C,A), h(D,B)$ intersection of coverage set (1,2,3,4,5,6,7,8)	P_1:$e(A,B) : -b(C,D), h(C,A)$ coverage set (1,2,3,4,5,6,7,8) $P_{C_3} : e(A,B) : -b(C,D), h(D,B)$ coverage set (1,2,3,4,5,6,7,8)

4.1.2 The Algorithm

This proposed algorithm is embedded inside Tabular CRIS-wEF, during specialization just after unifying the clauses. If two concept descriptors have the same head predicate and only on different literal they are unified to generate one or more specific concept descriptors. The body length of the generated clauses are one more than their parents' body length.

To incorporate background predicates, each argument of the predicate is generalized in all possible ways such that binding properties of the refined clause may have different binding properties from its parents. In that case, parents with the same binding properties are extracted from each refined clause.

In Tabular CRIS-wEF, coverage sets of generated clauses are stored in a hash table. After parents are extracted, coverage sets of these parents are retrieved from hash table and their intersection is calculated. Since the main algorithm generalizes the concept descriptors in all binding properties, missing coverage set in the hash is not an issue.

If the intersection size is higher than the support threshold value, the algorithm continues as in Tabular CRIS-wEF.

```
1:  for i = 0; i < pC.size() - 1; i++) do
2:    for j = i+1; j < pC.size(); j++ do
3:      if unifiable(pC[i], pC[j]) then
4:        tmp_pC = unify(pC[i], pC[j])
5:        for k = 0; tmp_pC.size(); k++ do
6:          parent[k] = getParent(k)
7:        end for
8:        support_set = set_intersect(parent)
9:        if support_set.size() ≥ min_set_size then
10:         pruneFurther(tmp_pC[k])
11:       end if
12:     end if
13:   end for
14: end for
```

Algorithm 2: Coverage Set Based Pruning

4.2 Cosine Similarity-based Pruning

In this part of the thesis work, we propose a pruning method by using similarity analysis of the arguments. As in the previous pruning method, this method is also embodied in Tabular CRIS-wEF [55, 53]. It is a two step algorithm. A preprocessing step is applied before running the ILP algorithm. In this step the terms for domain vector are collected, term-document count matrix is built and similarities of argument vectors are calculated. The second step takes place within the concept discovery algorithm, and it prunes the concept descriptors according to the similarities of *variables* in the clause. If two literals in a clause have same variable, cosine similarity of arguments of that variables are checked and pruned if their similarity is zero.

4.2.1 Motivation

Each predicate in an ILP system can be mapped to a table and arguments of the predicate are the columns that table. In kinship data sets (*daughter, elti, dunur*) arguments all belong to the same domain. An example of a relation-based data set is given in

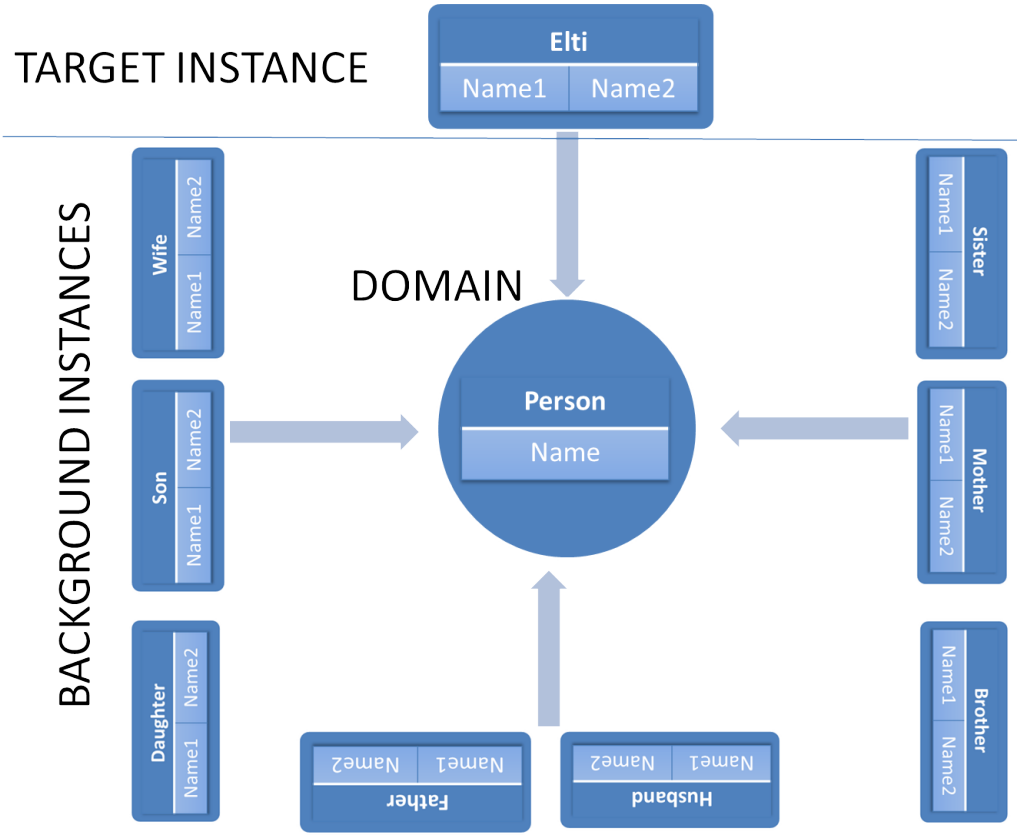

Figure 4.2: Elti data set domain

Figure 4.2.

As shown in the figure, all columns in target and background tables are connected to "name" column of Person table. Therefore all target and background column values are subsets of this column, Person.Name. Using Person.Name tuples as terms, and column values of target and background tables as documents, we are able to create our term-document count matrix where each count vector v $\in N^{|V|}$ and $|V|$ is the number of tuples in Person.Name, N is natural number that represents count of a term in the vector and a matrix size is $A \times |V|$ where A is the number of arguments that belong to the same domain.

In preprocessing step, if there are M different argument domains then M separate term-document count matrices should be created except for the ones that are supersets of exactly one argument in the whole data set. (E.g. only one argument in pte data set has $pte\_element$ domain, so there is no need to create a matrix for $pte\_element$ domain).

p_name	elti$name1	elti$name2	brother$name1	brother$name2	daughter$name1	daughter$name2	father$name1	father$name2
batuhan	0	0	1	1	0	0	0	1
bedriye	1	1	0	0	0	1	0	0
cagdas	0	0	1	0	0	0	0	1
cemile	2	2	0	0	0	2	0	0
dilber	0	0	0	1	2	0	0	1
dilek	0	0	0	0	2	0	0	1
erdem	0	0	1	0	0	0	0	1
esra	0	0	0	0	2	0	0	1

Figure 4.3: Term document count matrix for $Person.Name$ domain

Preparing the count matrix is a simple process, one query for each element in the domain is enough. For example, *elti* data set has eight background predicates and one target predicate. Since every predicate have two arguments and they all have the same domain (Person.Name), 9x2 =18 queries are enough to create a count matrix for a domain. Table 4.4 shows a query for calculating count vector for wife.Name2 and Figure 4.3 shows the document count matrix for Person.Name domain.

Table 4.4: Query for creating a count vector for wife.Name2

```
SELECT name2, COUNT(*)-1 vector FROM
    (SELECT name2 FROM wife
        UNION ALL
    SELECT name FROM person
    ) t
GROUP BY name2;
```

Count vectors in the proposed method are stored in a database table. Calculating the similarity of the argument vectors is completed before running CRIS and these similarities are stored in a table (see Figure 4.4). At the beginning of CRIS algorithm, cosine similarities are fetched from the database and stored in a hash table. During the specialization step, every newly refined concept descriptor checked as follows: if arguments of literals in a clause have the same variables and their cosine vector is zero, then it is pruned.

SQL query for calculating the support of a hypothesis in an Apriori-based ILP system is basically joins joins applied to tables on DBMS. Since a concept descriptor is pruned according to the result of its support query returned from DBMS, a similarity matrix based on the count vector can lead us estimating the result of the query without

```
1 •  SELECT * FROM
2    elti._cosine_score_person$name
3    order by value;
```

esult Grid | Filter Rows: | Export:

KEY1	KEY2	VALUE
elti$name2	father$name2	0
elti$name1	father$name2	0
elti$name2	daughter$name1	0
elti$name1	brother$name1	0
husband$name2	wife$name2	0
elti$name1	brother$name2	0
elti$name2	husband$name1	0
elti$name1	father$name1	0
son$name1	son$name2	0.117851
daughter$name1	daughter$name2	0.161164
brother$name1	son$name2	0.174131

Figure 4.4: Cosine score table for Elti data set

Table 4.5: Example of 3 pruned clauses

Pruned Clause	Description
elti(A,B):-daughter(**C**,A), father(**C**,D)	Pruned, $cosine(daughter.Name1, father.Name1) = 0$
elti(A,B):-brother(**C**,D), daughter(**C**,B)	Pruned, $cosine(brother.Name1, daughter.Name1) = 0$
elti(A,B):-brother(**C**,D), husband(D,A), daughter(**C**,A)	Pruned, $cosine(brother.Name1, daughter.Name1) = 0$

running it.

The arguments of $Elti$ data set subject to pruning based on the cosine score can be seen at Figure 4.5. It is interesting that with the proposed method, we can distinguish arguments that store people with different genders. E.g. Daughter.Name1 and Son.Name1 columns store female and male people records respectively. Consider a literal daughter(A,B) where A is daughter of B, thus A is female. Additionally, son(A,D) where A is son of D, thus A is male. If there is a concept descriptor has a conjunction of daughter.Name1 and son.Name1 with same variables, it should be pruned (e.g. elti(A,B):-daughter(C,A), son(C,A))). Cosine score of this arguments are zero, so such conjunction of literals are pruned before running the support query in DBMS.

In attribute-based data sets ($muta$, PTE, $eastbound$, $mesh$) most of the arguments have distinct domains so creating a single term-document matrix is not enough to

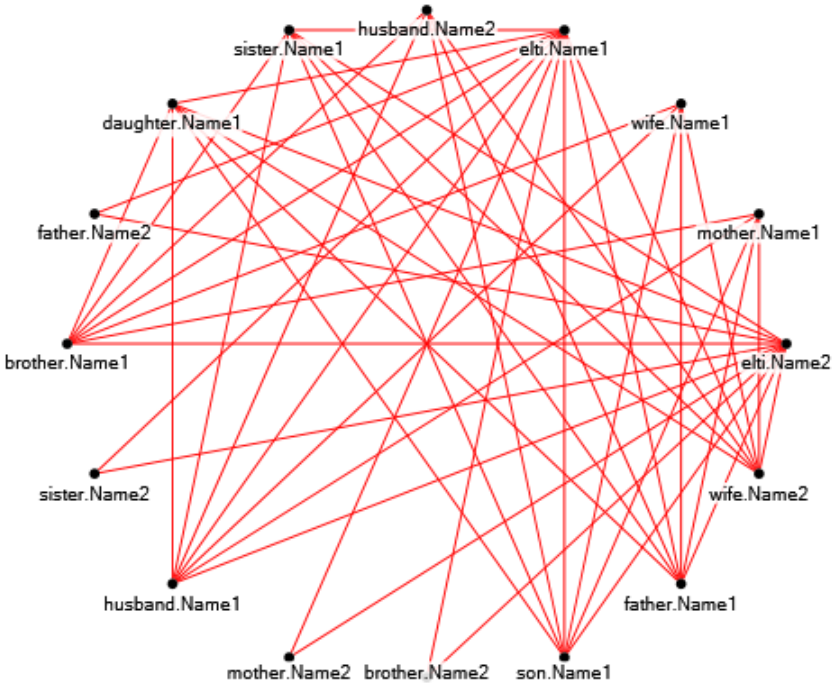

Figure 4.5: Arguments that have no relationship according to the cosine score

produce the similarity table. If more than one arguments are based on a domain, a count matrix and similarity table is generated for that domain. There is no need to generate a similarity table for single argument domains, their vector is assumed to be dissimilar from other vectors (so their cosine score with other arguments is set as 0). This assumption enables pruning the search spaces in a case where there is not enough dissimilar vectors in all domains of a data set. At worst case, where there is no dissimilar vectors to be pruned, this algorithm prunes clauses that has connected arguments (has same variables) but in different domains.

Experiments using different data sets has shown that, the algorithm prunes the search space up to 32% before running support queries. On the other hand, pruned queries have lower query costs that comes up with a lower time efficiency with the average of 5%.

4.2.2 Discussion

The arguments that are similar according to their cosine similarity is shown in Table 4.6. Although these results are not used for pruning, they may give us a shortcut to populate all hypothesis. Consider the two different hypothesis generated for elti data

Table 4.6: Arguments with Cosine Score=1

Argument1	Argument2	Cosine Score
elti.Name2	elti.Name1	1
husband.Name2	wife.Name1	1
mother.Name2	father.Name2	1
wife.Name2	husband.Name1	1

set:

S1:elti(A,B):-brother(C,D), wife(A,C), wife(B,D)
S2:elti(A,B):-brother(C,D), husband(C,A), husband(D,B)

We know that wife.Name2 and husband.Name1 are similar according to the cosine score Table 4.6. In addition, wife.Name1 and husband.Name2 are similar. We can safely generate the second solution using its similarity information. A transformation from wife(X,Y) to husband(Y,X) with the same binding properties generates the second solution. Although this methodology does not prevent generating all candidates to find a solution (is that not added to the proposed method), it enables populating more solution clauses from a single solution.

4.3 CRIS on Different DBMS Systems

4.3.1 MongoDB Implementation

In this section we investigate the feasibility of using NoSQL document oriented database system MongoDB as a storage for CRIS. By August of 2015, this NoSQL system took fourth place in ranking according to the worldwide popularity of databases. Being an open source database in addition to its popularity directs our attention to this database.

MongoDB is not intended to store relational tables. A huge difference of MongoDB from relational DBMSs is that, it does not support join operations between collections. Using embedded data model, related data is kept in denormalized form in a single document. A query over such a document allows us to get data that has already

been joined. As seen in Table 3.6, a document is just a row in collection. Storing multiple rows in a document is not a problem due to the flexible BSON format.

Table 4.7: Document Structure in a collection

{
"_id" : "<unique pattern of a clause>",
"clause" : [<array of all clauses in given pattern>],
"data" : [<array of data is placed here>]
}

Figure 4.6: A document in elti collection

We aim to store the clauses in embedded model, in a way that, we remove the need for joining collections to retrieve the support of a concept descriptor. The document

structure we used to store clauses and its data is given in Table 4.7. In MongoDB, "_id" field of a document should be unique in a collection. We rename the clause arguments according to their bindings in order to get a unique and generic id. All bounded arguments are renamed with "S" prefix and a number starting from 1. All unbounded arguments are renamed with "C" prefix and a number starting from 1. An example is given in Figure 4.7.

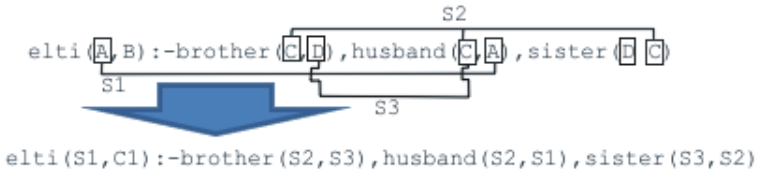

Figure 4.7: A general renaming of elti(A,B):-brother(C,D), husband(C,A), sister(D,C)

$Data$ field in the document stores the corresponding tuples of the clause in an array of documents. Each document in the array has fields names according to the generic argument names. Redundancy is reduced by storing only one of each connected arguments. E.g. for clause $elti(A, B) : -brother(C, D), husband(C, A), sister(D, C)$ given in the Figure 4.7, we store only S1, S2, S3 and C1 fields.

Support value of a single clause is queried using aggregation pipeline framework [42]. Figure 4.8 shows the query we used for calculating support. In the support query, documents are filtered using "match" operator. On the next stage, array is extracted using "unwind" and grouped according to the head literal arguments where unique head values remain. Final grouping gives the count of distinct values.

The proposed method operates in conjunction with algorithm Tabular CRIS-wEF [55, 53]. For constructing the search space, SQL queries are retrieved from MySQL database.

The algorithm performes poorly by using this data structure. In the nature of the ILP based systems, clauses are unified not to miss any indirect relations between predicates. This causes all possible combinations of clauses. Storing all these clauses in the embedded structure causes so much insertion time which makes it impossible to gain efficiency by aggregating from an embedded document.

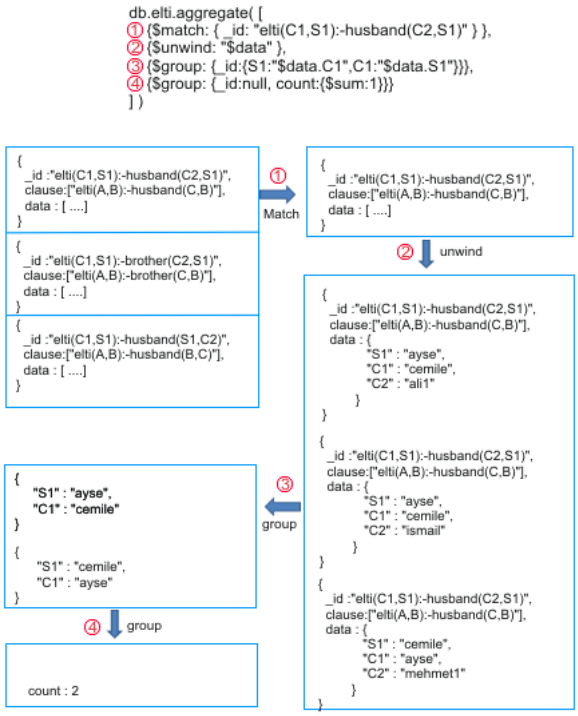

Figure 4.8: Aggregation pipeline for Calculating Support

4.3.2 Voltdb Implementation

The advantage of using a NewSQL system is to be able to use relational tables and querying the database using well-known SQL language in a clustered environment. Using relational tables makes it possible to convert the existing Tabular CRIS-wEF algorithm with minimum effort. Although structure of the database remains the same, some minor changes should be applied.

VoltDB is an in-memory database. This is another advantage of this system. Storing the target and background instances in the main memory enables us to test the current algorithm in traditional relational DBMS against an in-memory database. Data stored in RAM is volatile, after the system is shutdown all data is deleted from the memory. Some work has to be done to recover tables after restart of the system. VoltDB supports snapshots, which saves data to disks just as backups of relational systems. Snapshots can be taken either manually of automatically but this feature is available in paid version. Although creating a database is done by just running the VoltDB executable with $create$ parameter, after every start, DDL and DMLs of data set to be tested should be applied without snapshot support.

Another advantage of the system is supporting asynchronous calls. By using asynchronous instead of synchronous calls, client applications are able to submit all the jobs (queries) without waiting the result back. Submitting the job and retrieving the results are handled in different functions. In clustered environment, where there are more than one node, it allows to split the jobs to all clusters which seems to be appropriate while calculating support and confidence values serially. Performance analysis of this implementation is presented in Chapter 5.

Since it is a new born database, it has some restrictions which decreases with every new version. It was required to use stored procedures while interacting the DB with client applications, but it is not restricted any more. The pre-compiled procedures bring time performance by eliminating creation of explain plans before running the queries. Using stored procedures is not appropriable for CRIS. Because of the large search space, ILP systems produce a lot of possibilities of queries which are not known at run time and change by the given input parameters. For that reason, the experiments will be be based on adHoc queries instead of stored procedures.

4.3.3 Using MySQL "Memory" Storage Engine

ILP based concept discovery systems gain performance by the use of caching. In [12] 10.74% speed-up achieved on large data sets by the use of caching. Also in [55], memoization improves the time efficiency, for $muta$ data set, 14.27 speed-up is achieved.

In this part of the work, we mimic memoization by changing the storage engine of MySQL from InnoDB to $Memory$. Schema of the data sets and relations inside them all remain same.

Tables that use $Memory$ engine, store the data in the RAM of the server. The maximum size of MEMORY tables in MySQL is configurable, so only increasing a parameter to be able to store big data sets is enough.

The performance results by using Memory engine are given in Chapter 5.

CHAPTER 5

EXPERIMENTAL RESULTS

In this section, experimental results with the proposed methods are presented. First, we give details of the testing environment; server machine, DBMS software and data sets. Then we present the performance results of the proposed methods against Tabular CRIS-wEF.

5.1 The Environment

Tests are conducted on MySQL version 5.5.44-0ubuntu0.14.04.1. The DBMS resides on a machine with Core i7-2600K CPU processor and 7.8 GB RAM. C++ programming language is used for implementing the proposed methods. MySQL Connector is used to connect the client to the server. The client program and the DBMS software resides on the same host.

For minimizing the caching affect of the DBMS, SQL_NO_CACHE hint is used in the support and confidence queries. More information on caching configuration can be obtained in [1].

5.2 Data Sets

In the experiments, seven data sets from the literature are used. $Elti$ and $Dunur$ are relationship data sets. The former defines relationship between wifes of two brothers and the latter defines relationship between parents of a married wife and husband [34].

The arguments on target and background knowledge on these data sets all belong to the same domain "Person".

PTE data set includes nearly 300 classified drugs and their carcinogenicity of information whether they are carcinogenic or not [15]. The aim is to find patterns for identifying carcinogenicity of existing chemicals as well as new chemicals. Also to help chemists finding valuable patterns and accelerate their tests. The difference $PTENoAggr.$ and $PTE5Aggr.$ data sets is that, the latter contains additional aggregate predicates.

$Muta$ (mutagenesis) data set contains 188 classified drugs according to their mutagenicity [36]. Relationship between molecules and atoms and bonds are defined as background knowledge. We aim to find valuable patterns about mutagenicity of chemicals. $Muta\_small$ is a subset of $muta$ data set.

$Studentloan$ data set [58] contains 1000 student instances classified according to their obligation of repaying the loan.

In $eastbound$ data set information about cars; load of the car, shape of the car, shape of the load are given as background knowledge. With these kind of attributes of cars in a load of train , the aim is to determine whether the train belongs to east or not [40].

In $mesh$ data set [17] shape properties and neighborhood of elements from different dimensions are given as background knowledge. The aim is to analyze interaction between structures consist of these elements.

Dataset information and parameters used for the datasets are given in Table 5.1.

Table 5.1: Experimental parameters for each used data sets

Data_set	Num.relations	Num.facts	Min.sup.	Min.conf.	Length
Dunur	9	234	0.3	0.7	4
Elti	9	234	0.3	0.7	4
Muta_small	8	274	0.3	0.7	4
Muta	8	16,544	0.3	0.7	4
PTE_No_Aggr.	27	29,267	0.1	0.7	4
PTE_5_Aggr.	32	29,267	0.1	0.7	4
Student_loan	10	5,288	0.1	0.7	4
Mesh	26	1749	0.1	0.7	4
Eastbound	12	196	0.1	0.1	4

5.3 Proposed Method 1 (Utilizing Coverage Lists)

Time and query improvements of proposed method 1 are given in Table 5.2. "Filtering Queries" column shows the improvement ratio of $support$ and $confidence$ queries used for filtering the infrequent rules. The algorithm has no improvement on the count of rules and queries on $eastbound$ data set. This no gain affects time efficiency negatively. The highest gain in queries is achieved on $dunur$ data set hence with the greatest time increase. If we investigate the long running time data sets *PTE 5 Aggr.*, *Student Loan*, *Muta* and *Mesh* data sets, improvement on queries positively affects the running time of the algorithms. According to the table $mesh$ data set has the second highest percent time gain, which is much higher than expected, according to the query gain. The reason is that, Tabular CRIS-wEF and the proposed method uses hashing technique. Therefore improvement ratio results of "Filtering Queries" do not show the rate of actual count of running queries (**hash miss**) on DBMS.

Table 5.2: Improvements of Proposed Method 1

	Tabular CRIS-wEF			**Pruning by Coverage Lists**			**Improvement %**		
	Num. Rules	**Num. Queries**	**Time (mm:ss.sss)**	**Num. Rules**	**Num. Queries**	**Time (mm:ss.sss)**	**Rules**	**Filtering Queries**	**Time**
PTE 5 Aggr.	64322	237082	35:50.340	55729	228489	34:56.434	13.36	3.62	2.51
PTE No Aggr.	11166	43862	03:46.457	10121	42817	03:44.858	9.36	2.38	0.71
Elti	1741	5333	00:02.655	1540	5132	00:02.588	11.55	3.77	2.54
Dunur	1887	5807	00:02.086	1462	5382	00:01.862	22.52	7.32	10.72
Student Loan	305282	1441626	14:20.002	286270	1422614	14:07.700	6.23	1.32	1.43
Muta Small	6056	21044	00:10.774	5781	20769	00:10.664	4.54	1.31	1.02
Muta	62486	223644	34:04.099	56752	217910	33:40.651	9.18	2.56	1.15
Eastbound	7294	34654	00:04.091	7294	34654	00:04.248	0.00	0.00	-3.86
Mesh	56512	249084	00:27.302	51730	244302	00:25.537	8.46	1.92	6.46

Table 5.3: Improvement ratio of Num. of queries retrieved (**hash miss**) from DBMS for proposed method 1

	Support %	**(Filtering Queries)%**
PTE 5 Aggr.	14.15	6.94
PTE No Aggr.	10.55	5.96
Elti	13.19	7.70
Dunur	33.57	22.84
Student Loan	44.27	29.44
Muta Small	4.54	2.13
Muta	9.18	4.16
Eastbound	0.00	0.00
Mesh	45.21	38.19

Tabular CRIS-wEF stores $support$ queries and their results in hash table. Before a query is requested from DBMS, the algorithm first checks the query from the hash table, if it is in the hash (**hash hit**), it retrieves the result in the there, otherwise (**hash miss**) the query is requested from DBMS. The proposed algorithm does not prune any $confidence$ queries. Therefore time gain achieved by this algorithm is done by eliminating running $support$ queries. Table 5.3 lists the proportion actual running queries namely "hash miss" queries. Table shows that, $mesh$ data set has 38 % gain on running queries. This describes the time gain% of $mesh$ in Table 5.2.

5.4 Proposed Method 2 (Cosine Similarity Based Pruning)

Table 5.4: Improvements of Proposed Method 2

	Tabular CRIS-wEF			**Pruning by Coverage Lists**			**Improvement %**		
	Num. Rules	**Num. Queries**	**Time (mm:ss.sss)**	**Num. Rules**	**Num. Queries**	**Time (mm:ss.sss)**	**Rules**	**Filtering Queries**	**Time**
PTE 5 Aggr.	64322	237082	35:50.340	58503	231191	35:15.975	9.05	2.48	1.60
PTE No Aggr.	11166	43862	03:46.457	10328	43024	03:40.578	7.50	1.91	2.60
Elti	1741	5333	00:02.655	1422	4922	00:02.470	18.32	7.71	6.99
Dunur	1887	5807	00:02.086	1279	4607	00:01.783	32.22	20.66	14.54
Student Loan	305282	1441626	14:20.002	303565	1433041	14:22.295	0.56	0.60	-0.27
Muta Small	6056	21044	00:10.774	4910	19898	00:09.631	18.92	5.45	10.61
Muta	62486	223644	34:04.099	55477	216635	33:42.752	11.22	3.13	1.04
Eastbound	7294	34654	00:04.091	6805	32665	00:03.895	6.70	5.74	4.77
Mesh	56512	249084	00:27.302	54314	238982	00:27.314	3.89	4.06	-0.05

Table 5.4 shows time and query improvements of Proposed Method 2. As in method 1, this method also has highest gain in number of queries at $dunur$ data set.

Different from method 1, in method 2 $Studentloan$ and $mesh$ data sets have almost no time gain. If we investigate further, Table 5.5 shows that, these two data sets have minimum gains on number of running queries (hash miss). In addition, results show that for $mesh$ data set, 4.06 % query improvement in Table 5.4 decreases to 1.77 % in Table 5.5 which means that, high percentage of the queries that cause improvement are already stored in hash table.

For $eastbound$ data set 0 % query efficiency of method 1 increased to 5.74 % which shows that, two methods prune different sets of candidates.

Table 5.5: Improvement ratio of Num. of queries retrieved (**hash miss**) from DBMS for Proposed Method 2

	Support %	**Filtering Queries %**
PTE 5 Aggr.	9.57	4.70
PTE No Aggr.	8.46	4.78
Elti	19.42	11.33
Dunur	36.33	24.72
Student Loan	0.00	0.00
Muta Small	18.92	8.90
Muta	11.22	5.08
Eastbound	6.70	6.72
Mesh	2.10	1.77

As in Proposed Method 1, this method is also embedded in Tabular CRIS-wEF. Therefore only queries that are not stored in hash table are sent to DBMS (hash miss). Table 5.5 shows performance gain on *hash miss* queries. According to this table, this method has no improvement on *student loan* data set, that causes time worsening. Also for $mesh$ data set, 1.77 % improvement on queries does not overcome the time spent for the pruning method that results in increase of running time. In most of the data sets, "Filtering Queries" shown in Table 5.4 increases in "Filtering Queries %" in Table 5.5, this means high percentage of pruned queries are not in hash table.

5.5 Combination of Two Pruning Methods

Table 5.6: Comparison of improvement ratio of **Support** queries retrieved from DBMS (**hash miss**)

	Proposed Method 1	**Proposed Method 2**	**Proposed Method(1&2)**
PTE 5 Aggr.	14.15 %	9.57 %	23.24 %
PTE No Aggr.	10.55 %	8.46 %	18.93 %
Elti	13.19 %	19.42 %	25.92 %
Dunur	33.57 %	36.33 %	53.48 %
Student Loan	44.27 %	0.00 %	44.27 %
Muta Small	4.54 %	18.92 %	22.54 %
Muta	9.18 %	11.22 %	18.94 %
Eastbound	0.00 %	6.70 %	6.70 %
Mesh	45.21 %	2.10 %	46.37 %

In this section, combination two pruning methods are embedded into Tabular CRIS-wEF. This section presents the overall comparison of the proposed methods against

Tabular CRIS-wEf.

Table 5.6 shows the improvement of support queries for each data sets. For *PTE*, *student loan* and *mesh* data sets Proposed Method 1 performs pruning better then Proposed Method 2. On the remaining data sets, Proposed Method 2 performs better. It is also seen in the table that Proposed Method 2 is not appropriate for *student loan*, and Proposed Method 1 is not appropriate for *eastbound*. Hence there is no need to combine both methods for these data sets. For the remaining data sets, combination of two methods performs better.

Table 5.7: Comparison of improvement ratio of **filtering** queries retrieved from DBMS (**hash miss**)

	Proposed Method 1	**Proposed Method 2**	**Proposed Method(1&2)**
PTE 5 Aggr.	6.94 %	4.70 %	11.41 %
PTE No Aggr.	5.96 %	4.78 %	10.70 %
Elti	7.70 %	11.33 %	15.12 %
Dunur	22.84 %	24.72 %	36.38 %
Student Loan	29.44 %	0.00 %	29.44 %
Muta Small	2.13 %	8.90 %	10.60 %
Muta	4.16 %	5.08 %	8.58 %
Eastbound	0.00 %	6.72 %	6.72 %
Mesh	38.19 %	1.77 %	39.18 %

Table 5.7 shows filtering (support + confidence) queries that are retrieved from DBMS. All improvement ratios are decreased since both pruning methods does only *support* pruning. The distribution of rates is same with the previous table.

Table 5.8: Comparison of Improvement ratio of Running time

	Proposed Method 1	**Proposed Method 2**	**Proposed Method(1&2)**
PTE 5 Aggr.	2.51 %	1.60 %	4.13 %
PTE No Aggr.	0.71 %	2.60 %	3.45 %
Elti	2.54 %	6.99 %	7.47 %
Dunur	10.72 %	14.54 %	16.41 %
Student Loan	1.43 %	-0.27 %	1.02 %
Muta Small	1.02 %	10.61 %	10.64 %
Muta	1.15 %	1.04 %	2.05 %
Eastbound	-3.86 %	4.77 %	1.79 %
Mesh	6.46 %	-0.05 %	5.82 %

The running time improvements of each proposed method are shown in Table 5.8. For *student loan* and *eastbound* data sets, only the algorithm that prunes $support$

queries (Table 5.7 and Table 5.6) perform better. Also as seen in the table, for *mesh* data set, Proposed method 1 performs better than the combination. Proposed method 2 has almost no improvement for this data set. Gain in filtering queries (1.77%) shown in Table 5.7 does not overcome the algorithm cost hence does not improve the performance.

5.6 MongoDB

Because of the 16 MB restriction of document size, tests are performed at 4 data sets $muta\_small$, $dunur$, $elti$ and $eastbound$ those do not exceed the limitation. Average running time results of 10 runs for each data set is given in Table 5.9.

Results show that, querying from the embedded structure does not overcome time spent for storing the results in this structure. The problem is because of the chosen model for the ILP system. Tabular CRIS-wEF performs 2 times better then $dunur$, $elti$ and $eastbound$ data sets. For $muta\_small$ data set, the cost of MongoDB decreases because of the iteration count. Different from 3 other data sets those approach generating solutions in 1 iteration, $muta\_small$ achieves the goal in 2 epochs. That enables using previously generated embedded data in the collections, which decreases the loss.

Table 5.9: Comparison of running times of Tabular CRIS-wEF and MongoDB (mm:ss.sss)

	Running Time (mm:ss.sss)	
	Tabular CRIS-wEF	**MongoDB implementation**
muta_small	00:10.774	00:13.106
Dunur	00:02.086	00:04.456
elti	00:02.655	00:05.277
Eastbound	00:04.091	00:08.394

5.7 VoltDB

We will not compare the run time results of VoltDB against MySQL database. MySQL creates preferable explain plans, thus produce better timing results than VoltDB.

VoltDB performance even get worse while running complex queries that consist of more than tree tables in larger databases such as $pte5$, $muta$.

Order of tables in a query is a factor while generating plans in VoltDB. The queries are dynamically in the proposed methods, no heuristic is developed for the order of joins.

Some of the generated SQL that run on MySQL provides error in VoltDB. Conversion of some SQL queries needed. Also VoltDB does not allow joining two columns that have different domains. The proposed method "cosine similarity based pruning" eliminates such joins. Therefore we store actual running $support$ and $confidence$ queries of the method 2 and test them in this database.

Experiments are conducted on 3 virtual machines using docker [69] virtualization software on the same host. $sitesperhost$ parameter in VoltDB configuration defines the number of sites in a server machine. It should be set approximately 3/4 of the number of CPUs of the system. The server has 8 cores, but 3 hosts reside on the same machine. Therefore this parameter is set to 2 for each of the nodes and $2 \times 3 = 6$ in total. Synchronous and asynchronous calls of adHoc queries are tested on 1 to 3 node clusters. All tables are replicated to all nodes.

Table 5.10 shows average of 7 runs over given number of nodes. Results show that asynchronous calls perform better than synchronous calls to the database, even with a single node. Adding a node to the cluster increases the performance of async. calls. On the other hand, for 3 of the datasets performance of sync. calls decreases after adding nodes.

Table 5.10: VoltDB run timing results(mm:ss.sss)

	Node Count=1		**Node Count=2**		**Node Count=3**	
	sync.	**async.**	**sync.**	**async.**	**sync.**	**async.**
elti	00:16.693	00:13.110	00:17.271	00:08.351	00:20.820	00:06.214
dunur	00:09.153	00:07.409	00:01.954	00:00.482	00:02.211	00:00.754
muta_small	02:38.567	02:25.187	03:18.954	01:42.587	03:46.505	01:33.968
eastbound	00:02.557	00:00.856	00:02.763	00:00.614	00:04.214	00:00.519

5.8 MySQL Memory Storage Engine

With MySQL memory engine, data inside the table is stored in the memory, only the definition of the table is stored in the disk. Since database connection information in Tabular CRIS-wEF is given as input parameter, there is no need to change the structure of the developed code.

We created similar schema of data sets that only differs in "storage engine" of the tables, they were changed from $InnoDB$ to $MEMORY$. Since data is stored volatile and flushed after shutdown, a script to copy InnoDB tables to Memory engine is needed after a restart.

As seen at Table 5.11, by using $memory$ engine, performance gain is over 40 % for long running data set PTE. In addition, 25 % gain is achieved for $muta$ data set. These results show that, for long running data sets, creating a similar schema that differs only by "Engine" may help testing the algorithm in higher speeds.

Table 5.11: Tabular CRIS-wEF (MEMORY Storage Engine vs. InnoDB)

	InnoDB	**Memory Engine**	**Time gain %**
pte5Aggr	35:50.340	20:01.565	44.12
pte_noaggr	03:46.457	02:21.037	37.72
elti	00:02.655	00:02.227	16.12
Dunur	00:02.086	00:01.449	30.56
StudentLoan	14:20.002	05:34.432	61.11
muta_small	00:10.774	00:08.935	17.07
muta_01	34:04.099	25:15.102	25.88
Eastbound	00:04.091	00:03.764	7.98
Mesh	00:27.302	00:20.350	25.46

CHAPTER 6

CONCLUSION

In this thesis, we propose two pruning methods for an ILP system that uses relational tables. In the first pruning method we prune the search space by using intersection of coverage sets. In the second pruning method, cosine similarity of argument vectors of literals are used for pruning. Results show that, for *student loan* and *mesh* data sets only proposed method 1 improves the efficiency of the algorithm. On the other hand for *eastbound* data set only proposed method 2 provides efficiency. In addition, results show that these two methods prune mostly different concept descriptors. Therefore for the rest of the data sets, combination of two pruning methods produce more efficient results.

Results also show that gain in pruning does not reflect time efficiency. One of the reasons is that calculating support requires only one query, on the other hand calculating confidence requires two queries; one for numerator and one for denominator. Timing cost of confidence queries are much more than support queries. E.g. for long running data set *muta*, about 80 % of time spent for support+confidence queries belong to only confidence queries and for *pte* about 70 % belongs to confidence queries. This decreases the gain in time efficiency.

The data model chosen in MongoDB implementation cause high volume of data insertions which causes performance problems. In addition, this model is not appropriate for high cardinality of background instances. There may be much efficient BSON formats that does not produce redundant records but increase query performance. Graph methodologies may be more appropriate while applying ILP systems to NoSQL system.

By using MySQL Memory run time performance in all data sets increases. 44% running time improvement in large data set PTE also proves that most time consuming part of the Appriori-based ILP systems is querying and retrieving data from the DBMS.

As a future work, proposed method 2 may be extended in a way that, it is used as a reduction factor concept descriptors. In this thesis we prune the concept descriptors that have cosine score 0. By adding a minimum cosine score threshold value to the algorithm, it can also be used to prune additional concept descriptors whose cosine score is higher than 0 and below the threshold.

In addition, proposed method 2 may be extended in a way that, it is used while generating the the search lattice.

REFERENCES

[1] Query cache select options. https://dev.mysql.com/doc/refman/5.5/en/query-cache-in-select.html. Accessed: 2015-08-08.

[2] Newsql - scale-out sql database for the cloud | nuodb. http://www.nuodb.com/, 2015. [Online; accessed 17-August-2015].

[3] Rakesh Agrawal. *Fast Discovery of Association Rules.*, pages 307 – 328. Menlo Park, Calif.:, IBM Almaden Research Center, 1996.

[4] Inc Amazon Web Services. Aws | amazon dynamodb - nosql cloud database service. http://aws.amazon.com/dynamodb/, 2015. [Online; accessed 17-August-2015].

[5] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[6] Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Ramon, and Henk Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 2002.

[7] Ivan Bratko, Stephen Muggleton, and Alen Varsek. Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 385–388, 1994.

[8] Clustrix. Scale-out newsql database in the cloud | distributed sql db. http://www.clustrix.com/, 2015. [Online; accessed 17-August-2015].

[9] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[10] Vítor Santos Costa, Ashwin Srinivasan, and Rui Camacho. A note on two simple transformations for improving the efficiency of an ilp system. In *Inductive Logic Programming*, pages 225–242. Springer, 2000.

[11] Vítor Santos Costa, Ashwin Srinivasan, Rui Camacho, Hendrik Blockeel, Bart Demoen, Gerda Janssens, Jan Struyf, Henk Vandecasteele, and Wim Van Laer. Query transformations for improving the efficiency of ilp systems. *The Journal of Machine Learning Research*, 4:465–491, 2003.

[12] James Cussens. Part-of-speech tagging using progol. In *Inductive Logic Programming*, pages 93–108. Springer, 1997.

[13] Luc Dehaspe and Luc De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet familiarization workshop on statistics, machine learning and knowledge discovery in databases*, volume 1, page 5. Citeseer, 1995.

[14] Luc Dehaspe and Luc De Raedt. Mining association rules in multiple relations. In *Inductive Logic Programming*, pages 125–132. Springer, 1997.

[15] Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery*, 3(1):7–36, 1999.

[16] B Demoen, A Srinivasan, S Wrobel, and Luc DEHASPE. Frequent pattern discovery in first-order logic. 1998.

[17] Bojan Dolsak and Stephen Muggleton. The application of inductive logic programming to finite element mesh design. In *Inductive logic programming*. Citeseer, 1992.

[18] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. A survey in traditional information retrieval models. 2008.

[19] Lauren B Doyle. Information retrieval and processing. *Los Angeles, Calif.: Melville Publ. Co*, 681, 1975.

[20] Sašo Džeroski. Relational data mining. *Springer*, 2010.

[21] Sašo Dzeroski and Nada Lavrac. Inductive logic programming: Techniques and applications, 1994.

[22] Cao Feng. Inducing temporal fault diagnostic rules from a qualitative model. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 403–406, 2014.

[23] Nuno Fonseca, Rui Camacho, Fernando Silva, and V Santos Costa. Induction with april: A preliminary report. Technical report, Technical Report DCC-2003-02, Department of Computer Science, University of Porto, 2003.

[24] Nuno Fonseca, Vitor S Costa, Fernando Silva, and Rui Camacho. On the implementation of an ilp system with prolog. Technical report, Technical report, DCC-FC & LIACC, UP, 2003.

[25] Nuno Fonseca, Vitor Santos Costa, Fernando Silva, and Rui Camacho. Experimental evaluation of a caching technique for ilp. In *Progress in Artificial Intelligence*, pages 151–155. Springer, 2003.

[26] Nuno Fonseca, Fernando Silva, Vitor Santos Costa, Rui Camacho, et al. A pipelined data-parallel algorithm for ilp. In *Cluster Computing, 2005. IEEE International*, pages 1–10. IEEE, 2005.

[27] Zhiqiang Gao, Zhizheng Zhang, and Zhisheng Huang. Learning relations by path finding and simultaneous covering. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 5, pages 539–543. IEEE, 2009.

[28] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *Advances in information retrieval*, pages 345–359. Springer, 2005.

[29] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam AM Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):22, 2013.

[30] Franz Inc. Allegrograph rdfstore web 3.0's database. `http://franz.com/agraph/allegrograph/`, 2015. [Online; accessed 17-August-2015].

[31] Google Inc. Google cloud bigtable. `https://cloud.google.com/bigtable/`, 2015. [Online; accessed 17-August-2015].

[32] M Karthikeyan and P Aruna. Probability based document clustering and image clustering using content-based image retrieval. *Applied Soft Computing*, 13(2):959–966, 2013.

[33] Yusuf Kavurucu. *An ILP-based concept discovery system for multi-relational data mining*. PhD thesis, Middle East Technical University, Computer Engineering Department, 2009.

[34] Yusuf Kavurucu, Pinar Senkul, and Ismail Hakki Toroslu. Analyzing transitive rules on a hybrid concept discovery system. In *Hybrid Artificial Intelligence Systems*, pages 227–234. Springer, 2009.

[35] Ross D King, Stephen Muggleton, Richard A Lewis, and MJ Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the national academy of sciences*, 89(23):11322–11326, 1992.

[36] Ross D King, Stephen H Muggleton, Ashwin Srinivasan, and MJ Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996.

[37] Ross D King, Ashwin Srinivasan, and Luc Dehaspe. Warmr: a data mining tool for chemical data. *Journal of Computer-Aided Molecular Design*, 15(2):173–181, 2001.

[38] Arno Knobbe, Hendrik Blockeel, Arno Siebes, and Daniel van der Wallen. Multi-relational data mining. 1999.

[39] Richard Kufrin. Generating c4. 5 production rules in parallel. In *AAAI/IAAI*, pages 565–570. Citeseer, 1997.

[40] James Larson and Ryszard S Michalski. Inductive inference of vl decision rules. *ACM SIGART Bulletin*, (63):38–44, 1977.

[41] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[42] MongoDB. Aggregation Pipeline; MongoDB Manual 3.0.5. `http://docs.mongodb.org/manual/core/aggregation-pipeline/`, 2015. [Online; accessed 09-August-2015].

[43] MongoDB. MongoDB Documentation. `http://docs.mongodb.org/master/MongoDB-manual.pdf/`, 2015. [Online; accessed 05-August-2015].

[44] S Muggleton, R King, and M Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5(7):647–657, 1992.

[45] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[46] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the fifth international conference on machine learning*, pages 339–352, 1992.

[47] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[48] Stephen Muggleton, Cao Feng, et al. *Efficient induction of logic programs*. Citeseer, 1990.

[49] Alev Mutlu. *Improving scalability and efficiency of ILP-based and graph-based concept discovery systems*. PhD thesis, Middle East Technical University, Computer Engineering Department, 2013.

[50] Alev Mutlu, Mehmet Ali Berk, and Pinar Senkul. Improving the time efficiency of ilp-based multi-relational concept discovery with dynamic programming approach. In *Computer and Information Sciences*, pages 373–376. Springer, 2010.

[51] Alev Mutlu, Abdullah Dogan, and Pinar Karagoz. Utilizing coverage lists as a pruning mechanism for concept discovery. In *Information Sciences and Systems 2014*, pages 269–276. Springer, 2014.

[52] Alev Mutlu and Pinar Karagoz. A hybrid graph-based method for concept rule discovery. In *Data Warehousing and Knowledge Discovery*, pages 327–338. Springer, 2013.

[53] Alev Mutlu and Pinar Senkul. Improving hit ratio of ilp-based concept discovery system with memoization. *The Computer Journal*, 57(1):138–153, 2014.

[54] Alev Mutlu, Pinar Senkul, and Yusuf Kavurucu. Mpi-based parallelization for ilp-based multi-relational concept discovery. In *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, volume 1, pages 59–62. IEEE, 2011.

[55] Alev Mutlu, Pinar Senkul, and Yusuf Kavurucu. Improving the scalability of ilp-based multi-relational concept discovery system through parallelization. *Knowledge-Based Systems*, 27:352–368, 2012.

[56] Alev Mutlu, Pinar Senkul, and Yusuf Kavurucu. Improving the scalability of ilp-based multi-relational concept discovery system through parallelization. *Knowledge-Based Systems*, 27:352–368, 2012.

[57] Claire Nédellec, Céline Rouveirol, Hilde Adé, Francesco Bergadano, and Birgit Tausend. Declarative bias in ilp. *Advances in inductive logic programming*, 32:82–103, 1996.

[58] Michael J Pazzani, Clifford A Brunk, and Glenn Silverstein. A knowledge-intensive approach to learning relational concepts. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 432–436, 1991.

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

[60] J Ross Quinlan and R Mike Cameron-Jones. Foil a midterm report. In *Machine Learning ECML-93*, pages 1–20. Springer, 1993.

[61] Shivangi Raman, Vijay Kumar Chaurasiya, and Swaminathan Venkatesan. Performance comparison of various information retrieval models used in search engines. In *Communication, Information & Computing Technology (ICCICT), 2012 International Conference on*, pages 1–4. IEEE, 2012.

[62] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

[63] Balwinder Saini, Vikram Singh, and Satish Kumar. Information retrieval models and searching methodologies: Survey. *Information Retrieval*, 1(2), 2014.

[64] G Salton and M J McGill. Introduction to modern information retrieval. *Introduction to modern information retrieval*, 1983.

[65] Open Source. The apache cassandra project. http://cassandra.apache.org/, 2015. [Online; accessed 17-August-2015].

[66] Open Source. Apache couchdb. http://couchdb.apache.org/, 2015. [Online; accessed 17-August-2015].

[67] Open Source. Apache hbase™ home. http://hbase.apache.org/, 2015. [Online; accessed 17-August-2015].

[68] Open Source. Couchbase. http://www.couchbase.com/open-source, 2015. [Online; accessed 17-August-2015].

[69] Open Source. Docker - Build, Ship, and Run Any App, Anywhere. https://www.docker.com/, 2015. [Online; accessed 05-August-2015].

[70] Open Source. Look up a redis command. http://redis.io/, 2015. [Online; accessed 17-August-2015].

[71] Open Source. Memcachedb: A distributed key-value storage system designed for persistent. http://memcachedb.org/, 2015. [Online; accessed 17-August-2015].

[72] Open Source. Mongodb. https://www.mongodb.org/, 2015. [Online; accessed 17-August-2015].

[73] Open Source. Scalaris. http://scalaris.zib.de/, 2015. [Online; accessed 17-August-2015].

[74] Open Source. twitter/flockdb. https://github.com/twitter/flockdb, 2015. [Online; accessed 17-August-2015].

[75] Open Source. Voldemort. http://www.project-voldemort.com/voldemort, 2015. [Online; accessed 17-August-2015].

[76] Ashwin Srinivasan, Ross D King, and Douglas W Bristol. An assessment of ilp-assisted models for toxicology and the pte-3 experiment. In *Inductive Logic Programming*, pages 291–302. Springer, 1999.

[77] Ashwin Srinivasan, S Muggleton, and RD King. Comparing the use of background knowledge by inductive logic programming systems. In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 199–230. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[78] Jan Struyf and Hendrik Blockeel. Query optimization in inductive logic programming by reordering literals. In *Inductive Logic Programming*, pages 329–346. Springer, 2003.

[79] Mohameth-François Sy, Sylvie Ranwez, Jacky Montmain, Armelle Regnault, Michel Crampes, and Vincent Ranwez. User centered and ontology based information retrieval system for life sciences. *BMC bioinformatics*, 13(Suppl 1):S4, 2012.

[80] Orient Technologies. Orientdb multi-model nosql databaseorientdb multi-model nosql database. `http://orientdb.com/`, 2015. [Online; accessed 17-August-2015].

[81] Neo Technology. Neo4j, the world's leading graph database. `http://neo4j.com/`, 2015. [Online; accessed 17-August-2015].

[82] VoltDB. Using VoltDB. `http://docs.voltdb.com/UsingVoltDB/`, 2015. [Online; accessed 09-August-2015].

[83] John M Zelle and Raymond J Mooney. Learning semantic grammars with constructive inductive logic programming. In *AAAI*, pages 817–822, 1993.