COMENIUS UNIVERSITY
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

Peter Novák

Approximative Logic Programming for Data Mining

Diploma Thesis
Thesis Advisor: Doc. PhDr. Ján Šefránek, CSc.

BRATISLAVA 2002

By this I declare that I wrote this diploma thesis by oneself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

Bratislava, April 2002 Peter Novák

Approximative Logic Programming for Data Mining

Peter Novák, FMFI UK

E-mail address: `novak@kopernik.cc.fmph.uniba.sk`

FMFI UK, BRATISLAVA

Contents

CHAPTER 1

Introduction

Induction, *machine learning*, *approximative reasoning* and *data mining*. These are the main topics of this work. Nowadays we witness high research interest in these areas due to advances in the technology and the IT industry. Especially the field of the Data Mining is in the research focus of big IT companies, because it represents the area of possible revenues in the business world.

In the meantime, data about various aspects of people and the nature are stored in huge data stores and data warehouses. Few years ago these were only sources of concrete informations, but now, there's a need for powerful tools for analysis of these the data to mine additional implicit information content. For example statistical correlations can be extracted from huge collections of data. Such knowledge seems to be important in various fields, like marketing, planning, sociology, etc. Process of retrieving these nuggets of the knowledge is called the *knowledge discovery from databases*. Various methods of induction are used in this field. Usually a mix of statistical methods and fast database handling algorithms.

Nice results in the field of induction and machine learning were achieved by usage of neural networks and genetic programming. We agree with [**Sef 00**], that these are, so called, *blind* processes. Such techniques are non-transparent. In papers by Mannila and Muggleton ([**Mug 00**], [**JMW 96**] and [**Man 97b**]) we found serious basis for the symbolic approach to the induction and the data mining. On the ground of these methods we developed our own approach to data mining which was strongly inspired by methods described by Muggleton in his *Stochastic Logic Programming*, what is an approximative approach to the problem of induction.

In this work we describe *DMLP* (*Data Mining Logic Programming*). Despite the splendid name, we did our best to define simple and clear syntax and semantics for a kind of logic programming language. Without an ambition to develop an important result in the field of data mining, we believe that this work can offer interesting view on using logic programming for data mining from relational databases.

The text is organised as follows. The chapter 2 *Preliminaries* discusses basics of different kinds of logic programming and few basic definitions used in the core of this work. Basics of Logic Programming, Inductive Logic Programming and Stochastic Logic Programming are introduced in the first half of this chapter. In the second part basics of Data Mining and Knowledge Discovery in Databases. Condensed representations and Inductive Databases are discussed here, because approach to these two topics are primary interrests of this work.

The third chapter *Data Mining Logic Programs* is the core part of this work. In this chapter we define the syntax and the semantics of DMLP programs. It is supplemented by simple examples which demonstrate crucial ideas behind the text. It is written mainly as the set of definitions with few theorems and notes.

The chapter number 4 *Algorithms of DMLP induction* discusses the problem of induction over relational database and basic algorithms of induction of DMLP programs are introduced and their advantages and disadvantages are discussed here along with estimations of their complexities.

The fifth chapter *Usage of DMLP* deals with fields of possible usage of DMLP methods. The first section of this chapter discusses usage of DMLP programs as condensed representations in inductive databases. This part of the chapter is serious. The second part can be viewed as a speculation about other fields of usage of DMLP programs. It is just inspiration for further work, but it is not crucial for this paper. Therefore please consider it to be a not very serious part of this work.

Finally in the chapter 6 *Conclusions* we summarise this work and few open problems along with the inspiration for further work.

The last parts of this work is *Glossary of used abbreviations and symbols* and the bibliography. We just add that in the bibliography we cite only materials referenced in this work. For informative bibliography about research areas relative to this work we recommend to see [**Mug 00**], [**Man 97b**], [**Sef 00**] and [**Mug -**].

Reader who is familiar with basics of Inductive and Stochastic Logic Programming and Data Mining problems should skip directly to chapter 3 and do not read speculative part of chapter 5.

Sometimes the way of using capital letters at the beginning of names of well known methods may seem quite confusive. We use lower case first letters when we want to speak about the method in general (like mining the nuggets of knowledge from the database will be called *data mining*) and upper case first letters are used when we are speaking in the name of the method (*Data Mining* is the research field).

Finally we want to note that we did our best in usage of symbols and abbreviations in the same way as authors of papers, which are basic for this work, do. We hope that by this we make reading of this work easier.

CHAPTER 2

Preliminaries

Before we start with core topics of this work, we will need an introduction to areas which influence this work or on which topics discussed in the next chapters have some influence. These are theories of Logic Programming and Stochastic Logic Programming which are fundamental for understanding later results and we will provide an introduction to Knowledge Discovery in Databases and Data Mining.

2.1. Normal and Stochastic Logic Programs

Logic can be used as a programming language. This is the fundamental idea behind logic programming formulated in 1972 by Kowalski and Colmeraurer (according to [**Llo 87**]). It means that logic, usually used by mathematicians can quite easily be used as a powerful and expressive programming language for many areas where other formalisms fail. In previous decades we witnessed many theoretical and practical results in the field of Logic Programming which is merely fulfilling the fundamental idea given above. The best known and the most reputable language based on logic is PROLOG (PROgramming in LOGic), which has many clones such as Quintus PROLOG, micro-PROLOG, NU-PROLOG and many others.

In the next subsections we will shortly introduce syntax and semantics of logic programs, introduce version of Logic Programming used for induction from sets of examples (called Inductive Logic Programming) and finally familiarize with Stochastic Logic Programming, which is one of the fundamentals for our further work.

2.1.1. Logic Programming. In the following text we will briefly summarize syntax and semantics of logic programs and introduce standard terminology used in the next chapter which is the core part of the work. The next text will be extracted and reused from [**Mug 00**], because it fits our needs.

Syntax. Usually *variable* is denoted by a lower case letter v, x, y, z. Predicate and function symbols are denoted by lower case letters p, q and f, g, h. *Term* can be a variable or a function symbol immediately followed by a bracketed n-tuple of terms. Term with function symbol and no tuple as an argument is called *constant* and is written without brackets. Thus $f(g(x), h)$ is a term whenever f and g are function symbols, x is a variable and h is a constant.

A predicate symbol immediately followed by a bracketted n-tuple of terms is called an *atomic formula*, also called an *atom*. Atom can be negative, or positive. Negative atoms are prefixed by negation symbol $\neg$. Both a and $\neg a$ are *literals*, whenever a is an atom. From that we have also positive and negative literals.

Finite set of literals treated as a universally quantified disjunction is called a *clause*. *Horn clause* is a clause containing at most one positive literal and it is said to be *definite* if it contains exactly one positive literal. Definite clause for which all the variables in the head appear at least once in the body is called *range restricted*. For simplicity we use notation $A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$ for expression $\forall x_1, \ldots, \forall x_s(A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \vee \neg B_n)$, which denotes clause. Set of literals on the left side of the implication is called the *head* and set of literals on the right side is called the *body*. Clause with an empty head is called a *goal*. This is a standard convention in the field of Logic Programming.

Finite set of clauses is called a *clausal theory* and is treated as a conjunction of those clauses. Literals, clauses, clausal theories and True and False are called *well-formed formulas*. A well-formed formula is said to be *ground* if it contains no variables. *Horn theory* is a clausal theory containing only definite clauses. *Range restricted definite program* is a clausal theory in which all clauses are range restricted.

Semantics. Let $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ be a set of couples where each v_i is a variable and each t_i is a term and for no distinct i and j is v_i the same as v_j. Such a set θ is called *substitution*. θ is said to be *ground* when all t_i are ground. Let F be a well-formed formula or a term and θ be a substitution. Then $F\theta$ will be called an *instantiation* of F by θ and it is formed by replacing every occurence of variable v_i by the term t_i. By this, $F\theta$ is an *instance* of well-formed formula F. We say that clause C *θ-subsumes* clause D (written as $C \preceq D$) iff there exists a substitution θ such that $C\theta \subseteq D$.

A first order language L is a set of well-formed formulas which can be formed from a fixed and finite set of predicate symbols, function symbols and variables. A set of ground literals I is said to be an *L-interpretation* (or an *interpretation*) in case it contains either a or $\neg a$ for each ground atom a in L.

Let M be an interpretation and $C = h \leftarrow B$ be a definite clause in L. M is said to be an *L-model* (or *model*) of C iff for every ground instance $h' \leftarrow B'$ of C in L, $B' \subseteq M$ implies $h' \in M$. M is a model of Horn theory P whenever M is a model of each clause of P. P is said to be *satisfiable* if it has at least one model and *unsatisfiable* otherwise. Suppose L is chosen to be the smallest first order language involving at least one constant, a predicate symbol and a function symbol of Horn theory P. In this case interpretation is called a *Herbrand interpretation* of P and the ground atomic subset of L is called a *Herbrand base* of P. I is called a Herbrand model of Horn theory P when I is both Herbrand and a model of P. According to Herbrand theorem it is satisfiable iff it has a Herbrand model. Let F and G be well-formed formulas. We say that F entails G (we will write $F \models G$), iff every model of G is a model of F.

Proof of Logic Program. An inference rule $I = F \rightarrow G$ states that a well-formed formula F can be rewritten by a well-formed formula G. We write that $F \vdash_I G$ iff there exists a series of applications of I which transform F to G. I is said to be *sound* iff for each $F \vdash_I G$ always implies $F \models G$ and *complete* when $F \models G$ always implies $F \vdash G$. I is said to be *refutation complete* if I is complete with G restricted to False. The substitution θ is said to be the *unifier* of atoms a and a' whenever $a\theta = a'\theta$. μ is the *most general* *unifier* of a and a' if and only if for all unifiers γ of a and a' there exists a substitution δ such that $(a\mu)\delta = a\gamma$.

The *resolution inference rule* is as follows. $((C \setminus \{a\}) \cup (D \setminus \{\neg a'\}))\theta$ is said to be a *resolvent* of the clauses C and D whenever C and D have no common variables, $a \in C$, $\neg a' \in D$ and θ is the most general unifier of a and a'. Suppose P is a definite program and G is a goal. Resolution is *linear* when D is restricted to clauses in P and C is either G or the resolvent of another linear resolution. The resolvent of such linear resolution is another goal. Assuming that literals in clauses are ordered, a linear resolution is SLD when the literal chosen to resolve on is the first in C. An SLD refutation from P is a sequence of such SLD linear resolutions , which can be represented by $D_{P.G} = \langle G, C_1, \ldots, G_n \rangle$, where each C_i is in P and the last resolvent is empty clause ($False$). The answer substitution is $\theta_{P,G} = \theta_1\theta_2 \ldots \theta_n$, where each θ_i is the substitution corresponding with the resolution involving C_i in $D_{P,G}$. If P is a range restricted definite program P and a is a ground atom, it can be shown that $P \models a$ by showing that $\{P, \leftarrow a\} \vdash_{SLD} False$. We say that clausal theory P is consistent iff $P \not\vdash False$. Negation by failure inference rule says that $\{P, \leftarrow a\} \not\vdash_{SLD} False$ implies $P \vdash_{SLDNF} \neg a$.

2.1.2. Inductive Logic Programming.

This subsection is strongly inspired by chapter *Induction* in [**Sef 00**].

We can see Inductive Logic Programming as an approach to specify and solve problem of induction with using terms and tools of Logic Programming. When speaking about induction we mean a process of generalisation from a set of some examples. By such generalisation we construct a structure in which we will store information about these examples in general (i.e. there won't be recorded information about any particular example). Because we want to use mechanisms of Logic Programming to approach problem of induction, then the final structure which will be an output of an induction will be a logic program.

Let us have a set of *examples*. We will denote it as Δ. We can divide Δ into two subsets $\Delta = \Delta^+ \cup \Delta^-$. Δ^+ will be a set of positive examples (those which will be described by final induced logic program) and Δ^- will be the set of negative examples (i.e. those which must not be described by an induced logic program). As it was said, the final product of an induction has to be a logic program, or a set of hypothesis denoted by symbol Φ. By this induction we can repose on some background, or standard knowledge about examples which is known to us before induction and may help with it. We will denote it by symbol Γ.

Let Δ, Γ, Φ be sets ofexpressions of language L. We say that Φ *inductively follows* from Δ with background knowledge Γ iff

- $\Gamma \not\models \Delta$
- $\Gamma \cup \Delta$ is consistent
- $\Gamma \cup \Phi \models \Delta$
- $\Gamma \cup \Phi$ is consistent

Note that in our case all Γ, Δ, Φ are logic programs. If the first condition ($\Gamma \not\models \Delta$) would not be satisfied we have no reason to induce, because Γ sufficiently describes examples from Δ. Condition of consistency of $\Gamma \cup \Delta$ says that examples from Δ are not contradictive to background theory (i.e. there's no paradox in Δ according to Γ). Under condition $\Gamma \cup \Phi \models \Delta$ we mean that it is possible to deduce (infer) examples Δ from background theory Γ and induced logic program Φ. Finally condition on consistency of $\Gamma \cup \Phi$ says that induced hypothesis cannot be trivial. It means that it may not be possible to infer anything from Φ.

In fact there can be many possible induced logic programs, these can even be in contradiction with each other. This says that such induction is a non-monotonic process ([**Sef 00**]).

Now we are prepared to proceed to the theory of Inductive Logic Programming. As it was said above, we will use logic programs and techniques of Logic Programming to approach problem of induction. Syntax of Inductive Logic Programming is the same as syntax of usual logic programs. Therefore we can directly step to semantic specification of Inductive Logic Programming.

DEFINITION. Let $\Delta = \Delta^+ \cup \Delta^-$ be sets of positive and negative examples. Let Γ be a background theory and Λ be a non-consistent set of clauses. Then we say that Φ is a *correct set of hypothesis* iff:

- $\Gamma \cup \Delta \not\models \Lambda$
- $\Gamma \cup \Phi \cup \Delta^- \not\models \Lambda$
- $\Gamma \not\models \Delta^+$
- $\Gamma \cup \Phi \models \Delta^+$.

This definition is equivalent to definition of the problem of induction above. We will only note that negative examples can be seen as integrity constraints.

Generic algorithm. Generic algorithm 1 for problem of induction as it was given by Muggleton and de Raedt in [**MuR 94**] works as follows. We start from background theory and positive examples. Then we induce final logic program step-by-step by using rules of induction and advances from specific hyphotesis to more and more general ones. It is well-advised to order the space of hypothesis according to the relation of generalisation and specialisation. The whole computation is based on searching of the space of hypothesis. Inference rules can extend the set of hypothesis. Pruning can be based on relations of generalisation and specialisation. If positive examples do not follow from any hypothesis H (and theory Γ), then they do not follow from any hypothesis more special than H.

Stop criteria can be implemented in many different ways. Given generic algorithm is a generalisation of many possible strategies of computation of inductive generalisations.

Let us take a closer look to inference rules of induction. The simplest example of an inductive inference rule is θ-subsumption.

DEFINITION. Let C_1 and C_2 be clauses and θ be a substitution. Then we can inductively compute $C_1$from C_2 iff $C_1\theta \subseteq C_2$. We say that C_1 *θ-subsumes* C_2. We also say that

Algorithm 1 Generic algorithm of induction (according to [**MuR 94**])
input: set of inference rules R
output: set of hypothesis QH

```
$QH := \Gamma \cup \Delta^+$
repeat
  take $H$ from $QH$
  take inference rules from $R$ applicable on $H$
  apply inference rules on $H$ - let results of this step be hypothesis $H_1, \ldots, H_n$
  $QH := (QH \setminus \{H\}) \cup \{H_1, \ldots, H_n\}$
  prune $QH$
until $QH$ satisfies stop criteria
```

C_1 is more *general* than C_2 which is more *special* than C_1. If there is a substitution θ for which $C_1\theta \subseteq C_2$, we will write $C_1 \leq C_2$.

In fact there are many approaches to Inductive Logic Programming based on these generic ideas. We will not write more about them, they can be found in literature about ILP (try references in [**Mug 00**], [**Sef 00**] or [**MuR 94**]).

2.1.3. Stochastic Logic Programs. Stochastic Logic Programs were introduced by Stephen Muggleton in [**Mug 96**]. At first they were introduced as a way of lifting stochastic grammars to the level of first order logic programs. Later they have been shown to be a generalisation of Hidden Markov Models, stochastic context-free grammars and directed Bayes' nets. Next text will be an extraction and a compilation from [**Mug 00**].

Syntax. Stochastic Logic Program is a set of labelled clauses $g{:}C$, where g is a probability (i.e. number in the range $[0, 1]$) and C is a first-order range restricted definite clause. The subset S_p of clauses in S with predicate symbol p in the head is called *definition* of p. For each definition S_p the sum of probability labels π_p must be at most 1. S is said to be *complete* if $\pi_p = 1$ for each predicate symbol p and *incomplete* otherwise. $P(S)$ represents the definite logic program which we will have after removal of all probability labels from stochastic logic program S.

Proof for SLP. A *Stochastic SLD refutation* (called SSLD refutation) is a sequence $D_{S,G} = \langle 1{:}G, g_1{:}G_1, \ldots, g_n{:}C_n \rangle$ in which G is a goal, each $g_i{:}C_i \in S$ and $D_{P(S),G} = \langle G, C_1, \ldots, C_n \rangle$ is an SLD refutation from $P(S)$. SSLD refutation represents repeated application of SSLD inference rule. This takes a goal $g : C$ and a labelled clause $q : C$ and produces labelled goal $pq{:}R$, where R is the SLD resolvent of G and C. The answer probability of $D_{S,G}$ is $Q(D_{S,G}) = \prod_{i=1}^{n} g_i$. The incomplete probability of any ground atom a with respect to S is $Q(a|S) \leq Pr(a|S) \leq 1$, where $Pr(a|S)$ represents the conditional probability of a given S.

Semantics. On the ground of proof of Stochastic Logic Program we can introduce semantics of SLPs. Suppose L is a first order language and D_p is a probability distribution over the ground atoms of p in L. If I is a vector consisting of one such D_p for every predicate symbol $p \in L$ then I is called a *distributional L-interpretation*. If $a \in L$ is an atom with predicate symbol p and I is an interpretation then $I(a)$ is called probability of a according to D_p in I.

Suppose L is chosen to be the smallest first order language involving at least one constant, a predicate and function symbols of Horn theory $P(S)$. In this case the interpretation is called the distributional Herbrand interpretation of S.

DEFINITION. An interpretation M is a distributional L-model of Stochastic Logic Program S, iff $Q(a|S) \leq M(a)$ for each ground atom a in L.

Again, if M is a model of S and M is Herbrand with respect to S then M is a distributional Herbrand model of S.

Muggleton in [**Mug 00**] describes methods of inducing such Stochastic Logic Programs from the set of examples and background theory. Still this theory is not developed for use in the field of data mining. Few algorithms for induction of SLPs were implemented within CProgol4.5, which is a version of Progol and that is a clon of PROLOG for induction with use of logic programming.

Stochastic Logic Programs were shown to be useful and Muggleton used them in computational chemistry and bioinformatics to define distributions for sampling within Inductive Logic Programming.

2.2. Knowledge Discovery in Databases and Data Mining

Research field of Knowledge Discovery in Databases, often called Data Mining, received a lot of attention in the last few years. This is because of a need of industry not only to store and manage the data about various domains, but to use them as good as it gets. Strong attention is dedicated to get interesting implicit information, which we are able to calculate from the database, but these are not explicitly stored there. Usually mix of statistics and computational tools and techniques is used. Data mining aims at the discovery of useful information from large collections of data. This knowledge can be retrieved from database in the form of rules describing properties of data, frequently occuring patterns or clusters of objects, etc.

Here we will give a brief introduction to fundaments of data mining and process of Knowledge Discovery in Databases. We will not go into deep results, because this research field is quite wide.

2.2.1. The KDD process. As it was said above, the goal of knowledge discovery is to obtain useful nuggets of knowledge from large collections of data. As it is obvious, such task is inherently interactive and iterative. A user of KDD system has to have understanding of the domain of the data in order to select the right subsets of data and good criteria of estimation whether given discovered pattern is interesting enough or not. Because of the fact that this knowledge can only hardly (if ever) be articulated and expressed by an artificial system, KDD systems will usually be just semiautomatic tools.

Therefore knowledge discovery from large databases can be seen as a process containing several steps. We will give here steps which are formulated in [**Man 97b**] by Heikki Mannila. These are:

(1) understanding the domain
(2) preparing the data sets

(3) discovering patterns (data mining step)
(4) postprocessing of discovered patterns, and finally
(5) putting the result into use.

As Mannila claims, the KDD process is neccessarily iterative. The results of data mining steps can show that some changes should be made to data set formation step, postprocessing of patterns can cause user to look for some slightly modified types of patterns, etc. One of the important research topics in KDD is an efficient support of such iterations.

Output patterns. Above, we mentioned that the task of knowledge discovery, or data mining is to find frequent patterns occuring in the dataset of our interrest. Now we have to explain what we mean under such *interesting patterns*. The character of a frequent pattern depends on the application domain of our data mining task.

In [**Man 97b**], Mannila discusses few basic instances of data mining problem and for each of them we have different type of frequent pattern occuring in the dataset. We will give here a brief overview of two most important ones (as it seems to us).

Given a schema $R = \{A_1, \ldots, A_n\}$ of attributes with domain $\{0, 1\}$, and a relation r over R, then an association rule about r is an expression of the form $X \Rightarrow B$, where $X \subseteq B$ and $B \in R \setminus X$. The intuitive meaning of the rule is that if a row of the matrix r has 1 in each column of X, then the row tends to have a 1 also in column B. Given $W \subseteq R$, we denote by $s(W, r)$ the *frequency* of W in r: the fraction of rows of r that have 1 in each column of W. The *frequency* of the rule $X \Rightarrow B$ in r is defined to be $s(X \cup \{B\}, r)$, and the *confidence* of the rule is $\frac{s(X \cup \{B\}, r)}{s(X, r)}$.

In the discovery of association rules, the task is to find all rules $X \Rightarrow B$ such that the frequency of the rule is at least a given treshold σ and the confidence of the rule is at least another treshold θ.

As Mannila claims (together with the rest of DM authors), in large retailing applications the number of rows might be 10^6, or even 10^8, and the number of columns around 5000. The frequency treshold σ is typically around $10^{-2} - 10^{-4}$. The confidence treshold θ can be anything from 0 to 1. From large databases one can obtain hundreds of thousands of association rules.

Generic algorithms for search for appropriate association rules are based on the idea of frequent sets, where *frequent set* is a subset $X \subseteq R$ for which we have that $s(X, r) \geq \sigma$. Once all frequent sets of dataset r are known, finding the association rules is a straightforward task. We only need to verify whether confidence of each rule that is possible to construct from given frequent set is sufficiently high.

Second instance of data mining problem discussed in this work, will be finding episodes from event sequences. Let us consider a sequence of events with timestamps of their occurence. As Mannila in [**Man 97b**] claims, such data are routinely collected usually in industry, telecommunication networks, process monitoring, epidemiology, etc.

Let us have the a set of all event types. An *episode* is a partially ordered set of events. An episode might be for example a structure representing the statement that events A and B occur before an event of type C.

Algorithms that search for frequent episodes are based on locating episodes of size 1 at first, then use them to generate episodes of size 2 and so on.

In this work we will pay special attention to the first instance of data mining problem. We will consider only manipulations with association rules, because these are basic patterns occuring in the relational database.

2.2.2. Data Mining as Selective Theory Extraction. Although the task of data mining is a synonymum for knowledge discovery process we will treat it as a search for frequent and interesting patterns in the large database. As Tomasz Imielinsky in [**Imi 95**] noted, the situation in data mining can be compared to the status of database processing in the 1960's: one had to write a separate application program for each query. Effort to develop a strong and sufficient theory in the background of data mining can be seen in the last few years. There is a need for theory, which will be able to express KDD queries and results of such computation. Steps towards this were taken in [**JMW 96**]. Mannila in this work proposes the use of probabilistic logic developed by Bacchus in [**Bac 90**]. He introduces simple theory based on arguing that

(1) the task of data mining can be seen as a problem of extracting the interesting part of the logical theory of a model; and
(2) the theory of a model should be formulated in a logic which is able to express quantitative knowledge and approximate truth[1].

In the next paragraphs we will briefly introduce Mannila's theory.

He considers a database over a single relation schema $R = \{A_1, \ldots, A_n\}$ of attributes A_i. Each attribute has a domain D_i. A database r over R is a set of tuples $t = (t[A_1], \ldots, t[A_n])$, where $\forall i : t[A_i] \in D_i$. Tuple relational calculus is constructed from variables s, t ranging over tuples in the database, constant symbols for every element of D_i, symbols for the functions and relations defined on D_i and the attribute symbols $A_1, \ldots, A_n$.

Terms are built simply by rules: every constant symbol for an element of D_i is a term of sort D_i. If $\sigma_1, \ldots, \sigma_m$ are terms and f is an m-ary function symbol for D_i, then $f(\sigma_1, \ldots, \sigma_m)$ is also a term of sort D_i. If s is a variable and A_i is an attribute symbol, then $s[A_i]$ is a term of sort D_i.

Atomic formulas are either of the form $T(\sigma_1, \ldots, \sigma_m)$, where each σ_j is a term of sort D_i and T is an m-ary relational symbol on D_i, or of the form $\sigma = \tau$ where σ and τ are terms of sort D_i.

By closing the language defined so far under boolean connectives ($\wedge$ and $\neg$) and quantification ($\exists$ and $\forall$) over the tuple variables one obtains the usual tuple relational calculus.

Afterwards this language is extended by constructs which allow it to express statements about approximate truth of formulas. In fact an *error term* is defined as either a constant symbol, q of some domain $\mathcal{Q}$ (e.g. $\mathcal{Q} = \mathbb{R} \cap [0, 1]$), or an expression of the form $G(\chi(s)|\psi(s))$, where $\chi(s)$ and $\psi(s)$ are formulas in tuple relational calculus whose free variables are among $s = (s_1, \ldots, s_k)$.

[1]Under the term *model* we mean model of logical theory.

An *atomic formula* is any expression of the form $\eta \leq \delta$ or $\eta = \delta$, with error terms η and δ. The final representation language is obtained by closing the union of tuple relational calculus and the set of atomic error formulas under boolean connectives and quantification. We will give an example of an expressions written in this language and discuss their advantages and disadvantages later.

For an example of such expression let us have relational schema R, set of columns (attributes) X, attribute B and treshold values for frequency σ and confidence θ. Then a formula

$$G(t[B] = 1 | \bigwedge_{A \in X} (t[A] = 1)) \geq \theta$$

means that the association rule $X \Rightarrow B$ has confidence θ. Similarly the formula

$$G(\bigwedge_{A \in X} (t[A] = 1) \wedge (t[B] = 1)) \geq \sigma$$

represents the fact that the rule $X \Rightarrow B$ has frequency σ[2].

Semantics of such rules is defined through *error measures*. Intuition behind this term is that an error measure represents a "degree of falsity" of χ given that ψ is true.

Mannila defines an error measure for R in [**JMW 96**] as any function g which assigns to every triple $(r, \chi(s), \psi(s))$, where r is a database over R and $\chi(s)$ and $\psi(s)$ are formulas of tuple relational calculus, a value $q \in R \cap [0, 1]$ such that

(1) $g(r, \chi(s), \psi(s)) = 0$ if $r \models \psi(s) \rightarrow \chi(s)$, and $r \models \exists s : \psi(s)$
(2) $g(r, \chi(s), \psi(s)) = 1$ if $r \models \nexists s : \psi(s)$
(3) $g(r, \chi(s), \psi(s)) \leq g(r, \chi'(s), \psi'(s))$, if $\models \psi(s) \leftrightarrow \psi'(s)$ and $\models \chi'(s) \rightarrow \chi(s)$.

Given an error measure g, the semantics of an error term $G(\chi(s)|\psi(s))$ in r is the value $g(r, \chi(s), \psi(s))$ and the relation $r \models_g \varphi$ is straightforwardly defined.

As it is obvious, many functions will meet requirements on error measure, even those which have quite bad properties. Mannila does not take these "bad" functions into account.

Generic data mining algorithm. In the next part we show the generic data mining algorithm for finding all frequent patterns (in general meaning of pattern). This algorithm (although naive) can be adapted to instances of data mining problem described above.

We will not discuss algorithm 2 more, because we will adapt this algorithm for our purposes later and it will be sufficiently discussed in chapter 4.

2.2.3. Condensed represenations. As it was said above, data mining is an iterative process. Outputs of one data mining step usually influence inputs into the second one. As an implication of that we can find out that similar data mining queries have to be evaluated many times. This is quite inefficient effect. This gives rise to the concept of *condensed representation*. There are two issues, which are solved by this concept.

The first is evaluation of similar queries faster than by looking at each of them individually. Ideally this has to be possible in as few passes through the database as possible.

And the second problem is, how to evaluate queries from a query class without looking at the whole data set.

[2]Note that an expression of the form $G(A|B)$ means conditional probability of A given that B holds.

Algorithm 2 FFP - Finding all frequent patterns (as it is introduced by Mannila in [**Man 97b**]).
Assume that there is an ordering $<$ defined between the patterns of $\mathcal{P}$.

```
$\mathcal{C} := \{p \in \mathcal{P} | \forall q \in \mathcal{P} : p < q\}$;
$\mathcal{F} := \emptyset$;
**while** $\mathcal{C} \neq \emptyset$ **do**
  **for** each $p \in \mathcal{C}$ **do**
    find the number of occurences of $p$ in $r$;
    $\mathcal{F} := \mathcal{F} \cup \{p \in \mathcal{C} | p$ is sufficiently frequent in $r\}$;
    $\mathcal{C} := \{p \in \mathcal{P} |$ all $q \in \mathcal{P}$ with $q < p$ have been considered already and it is possible that $p$ is frequent$\}$;
  **end for**
**end while**
output $\mathcal{F}$;
```

From the above mentioned it follows that if we have a class of structures $\mathcal{D}$, a data collection $r \in \mathcal{D}$, and a class of patterns $\mathcal{P}$, a *condensed representation* for r and $\mathcal{P}$ is a data structure that makes it possible to answer queries of the form *"How many times does $p \in \mathcal{P}$ occur in the database r?"* approximately correctly and more efficiently than by looking at the database r itself.

In literature, one can find the following simple example of a structure fulfilling requirements on condensed representation: a sample from the data set. By counting occurencies of given pattern in the sample one gets an approximation of number of occurencies in the original data set.

Few other examples were given and studied in literature. Still, according to our knowledge, none of them was good enough to be widely agreed upon as a good approach to this problem. Primary aim of this work is to propose a structure which seems to us to be a good candidate on condensed representation.

2.2.4. Inductive databases. According to what was said above about KDD as an iterative process, there is another aspect of this fact. User of an KDD system often wants to cross the boundary between data and structures which were computed as an output of previous data mining steps. For example, the user might want to view outputs of such KDD process, select some of computed patterns, look at the exceptions of these patterns, form a set of patterns describing these exceptions, etc. To make such moves between the data and the output patterns Mannila proposes the following principle:

KDD queries have to satisfy closure property: the result of a KDD query should be an object of a similar type than the arguments.

Relational databases satisfy this. Answers to SQL queries over relations are relations again. These requirements partially motivate the term of *inductive database*.

Framework of *inductive databases* introduced in [**Man 97b**] and elsewhere in the DM literature is focused on usage of association rules as an output of KDD query. The core idea is based on the database consisting of the raw data and some more or less inductivelly obtained rules. The term *inductive database* should be compared with the term *deductive database*. While deductive databases use simple form of deduction to augment fact databases to contain a potentially infinite set of derived or deduced facts, the goal of inductive database is that in addition to the facts, the database will contain a potentially infinite set of induced rules (or some form of more general information structure)

As Mannila claims in [**Man 97b**], developing condensed representations for various classes of paterns seems to be a promising way of implementing inductive databases and more generally improving the effectiveness of data mining algorithms. Whether this approach is generally useful is still open.

2.2.5. Open problems. In the last years we can see fast development in the field of data mining. As Agrawal mentioned in [**Agr 02**], data mining showed a strong potential for the future, still while this is a fast evolving research field, there are many open problems. One of the most important open problems seems to be a development of a general theory of data mining. Sometimes this is called the theory of inductive databases. This is tightly connected with development and usage of the concept of condensed representations. This work focuses on this research problem.

There are also other problems or issues concerning KDD systems and query languages. Architecture of KDD systems, support of KDD process from the side of the database management systems, development of language suitable to express KDD queries, algorithmic problems, etc. These are hot issues, what can be seen on talks on conferences and workshops about knowledge discovery and data mining.

CHAPTER 3

Data Mining Logic Programs

In the previous chapter we briefly described research fields which have influence on this work or motivation for it. In the core of this work, which are chapters 3 and 4, we will propose framework designed to solve some of the problems and issues mentioned in the previous chapter. We will introduce original framework which aims to attack the problem of condensed representations and through this also to influence the concept of induction database. We will adapt mechanisms developed by Stephen Muggleton for Stochastic Logic Programming and principles of logic programming to develop a structure able to serve as a condensed representation.

3.1. Introduction

As it was introduced above, this chapter will introduce syntax and semantics of *Data Mining Logic Programs* (DMLP). This kind of logic programs was strongly inspired by Stephen Muggleton's *Stochastic Logic Programs* ([**Mug 00**]), which we saw as a good example of probabilistic logic programs and wanted to use them (or to inspire ourselves) in the field of data mining. This sort of logic programs was chosen because of its well defined syntactic and semantic properties which we considered to be quite similar to requirements of the language used in the data mining field ([**JMW 96**]).

In 3.1.1 we will introduce first steps to syntax and semantics of DMLP to help understand sections 3.2 and 3.3, which contain the core of this work.

3.1.1. Elementary description of DMLP. We will define DMLP as a tool which can help us to solve the following problem. Let us have the relational table over finite domains (for simplicity we will use binary data) and we would like to construct an object which describes the database as good as it gets. Afterwards we can "ask" about properties of the database and we will be able to compute an answer with help of our description, without looking into the database (i.e. answering algorithm does not need to walk through each row of the database to compute an answer). Of course, such an object will not be any "oracle". It will be able to help us compute only appropriate type of questions. Our aim will be to build such mechanism for questions relative to statistic properties of the database (e.g. "How many percent of rows have value 1 in column A?"). As it was already mentioned, we were strongly inspired by SLPs and on the basis of Logic Programming we built our DMLP.

Data Mining Logic Program is a set of clauses labeled by probabilities with which they hold in the given relational database. For simplicity, we will use clauses only with one variable. From this we have that a clause will be able to express only the sentence about one row of the database. These clauses will be also restriced to express only sentences about the database. This requirement leads us to use predicates with fixed arity.

Development of semantics of our logic programs will be strongly inspired by mechanism of resolvence (widely described in literature about LP and introduced in the chapter 2). In fact, we will use very similar mechanism as Muggleton does in [**Mug 00**] to compute distributional model of SLPs. For this purpose we will define an evaluation function, which will be able to compute this distributional model from chain of clauses produced by SLD refutation (for explanation see [**Sef 00**], [**Llo 87**]).

3.2. Syntax

3.2.1. DMLP syntax.

DEFINITION 3.2.1. Let $R = (A_1, \dots, A_n)$ be a *relational schema* consisting of attributes named by symbols A_i. Each *attribute* A_i is defined over finite domain D_i. *Database* r over the relational schema R is a set of *tuples* $x = (x[A_1], \dots, x[A_n])$, where $\forall i : x[A_i] \in D_i$.

DEFINITION 3.2.2. *Variable* ranges only over tuples of the database. *Constant symbols* are defined for each element of D_i $\forall i \in 1 \dots n$ and we define also special constant symbols for real numbers from $\mathbb{R}$ used as *labels*. We define *predicate symbols* to operate only over tuples from the database (or variables).

NOTE 3.2.3. In the next text we will use the following convention for denotation of symbols in it. Variables will be denoted by letters u, v, x, y, z. Constants will be denoted by letters a, b, c, predicates by letters p, q. Database will be denoted by character r, relational schema by R and real number constants by g. All symbols can be subscripted or upper scripted. For indexes we will use symbols i, j, k, l, m, n. Clauses will be denoted by upper case letters C, D and sets of literals by upper case letters A, B. Sets of clauses will be denoted by upper case Greek letters. If these rules will be broken somewhere in the text it will be explained immediately. Special functions will be denoted by special characters ad hoc according to our needs.

DEFINITION 3.2.4. Let r be a database over a relational schema $R = (A_1, \dots, A_n)$ of attributes $A_1, \dots, A_n$ over finite domains $D_1, \dots, D_n$ and let x be variable ranging over the database r.

Terms are built as follows:

- Every constant symbol for an element of D_i is a term of sort D_i (these terms will be called *ground*).
- If x is a variable, then $x[A_i]$ is a term of sort D_i.

We define *well-formed formulas* inductively as follows:

- If p is an n-ary predicate symbol and $\sigma_1, \dots, \sigma_n$ are terms then $p(\sigma_1, \dots, \sigma_n)$ is a *well-formed formula* (*atomic formula*, *atom*).
- If σ_1 and σ_2 are terms of sort D_i then $\sigma_1 = \sigma_2$ is also *well-formed formula* (*atom*), we call it also *basic equality.*

- If F and G are formulas, then also $(\neg F)$, $(F \wedge G)$, $(F \vee G)$ are *well-formed formulas.*
- *Literal* is an atom or negation of an atom.
- Well-formed formula is said to be *ground* iff it contains no variables.

We define *clause* as a formula consisting of a finite set of literals connected by disjunction of the form $\forall x_1, \ldots, \forall x_s(L_1 \vee \cdots \vee L_m)$, where $L_1, \ldots, L_m$ are literals and $x_1, \ldots, x_s$ are all variables occurring in $L_1 \vee \cdots \vee L_m$.

NOTE 3.2.5. As it is usual, we denote the clause $\forall x_1, \ldots, \forall x_s(A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \vee \neg B_n)$ as $A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$. We say that the left side of the clause is called the *head* and the right side is called the *body*. All variables are assumed to be universally quantified, commas in the body $B_1, \ldots, B_n$ denote conjunctions and commas in the head $A_1, \ldots, A_k$ denote disjunctions.

DEFINITION 3.2.6. *Horn clauses* are clauses containing at most most one positive literal (at most one atom in the head). *Definite clause* is a clause containing exactly one positive literal. A non-definite Horn clause is called a *goal*. A definite clause for which all the variables in the head appear at least once in the body is called *range restricted*. We say that a clause is *recursive* iff the predicate, which occurs in the head of the clause, appears also in the body of the clause.

NOTE 3.2.7. Definitions given above were already mentioned in the chapter 2. We defined some of already known terms again to hold the definition of syntax of DMLP consistent.

In the next text we will use only *goals* and *range restricted definite clauses* with one variable.

NOTE 3.2.8. Set of clauses is treated as a conjunction of clauses (because well-formed formulas are inductively closed on conjunction, disjunction and negation, we can say that also a set of clauses is a well formed formula). We will use term *formula*, instead of longer *well-formed formula*.

DEFINITION 3.2.9. Clause in the form of g:C, where $g \in \mathbb{R} \cap [0, 1]$ and C is a range restricted definite clause or a goal, is called *DMLP clause*.

DEFINITION 3.2.10. *DMLP* (*Data Mining Logic Program*) Φ is a set of DMLP clauses g:C, where g is a value of the probability of Horn clause C. The subset S_p of clauses containing predicate symbol p in the heads, is called the *DMLP definition of* p.

NOTE 3.2.11. Let p be the predicate symbol and Φ be the DMLP program. We will write $p \in_{def} \Phi$ when Φ contains the definition of p. This is a syntactic notation.

DEFINITION 3.2.12. We say that $\Delta = \{p_r(a) \leftarrow |\forall a \in r\}$ is a set of *DMLP examples* and the predicate p_r used in such set is said to be *associated* with the database r. That is the database r expressed according to syntax of DMLP. We say that Δ is *associated* with r.

NOTE 3.2.13. From this point on, wherever we will mention the set of DMLP examples Δ we mean that there exists a relational database r associated with Δ. It means that the defitniion of r will be omitted wherever it will be possible. This only serves to shorten the text size.

DEFINITION 3.2.14. Let Φ be a DMLP program. Let Γ be a set of range restricted definite clauses defined according to syntax rules given above. We say that Γ is *DMLP background theory of* Φ when for each predicate symbol p, $p \in_{def} \Phi \implies p \in_{def} \Gamma$ and body of each clause $C \in \Gamma$ contains formula in the form of $p_r(x)$, where p_r is the predicate associated with the database r and x is a variable symbol used in the head of the clause C.

Background theory Γ will serve us as the set of definitions of predicates (tools) which can be used to describe relational database r. We will treat clauses of Γ as if their body does not contain member of the form $p_r(x)$. We will implicitly assume that each clause of Γ contains such formula.

As it was mentioned in 3.1.1, Data Mining Logic Program Φ will be that object which will help us compute answers to questions about statistical properties of given database r. Computation will run only over the program Φ without using the database r. By this we will be able to throw away the whole database r and manipulate only with Φ.

EXAMPLE 3.2.15. **Syntax**
Database r:

A_1	A_2	A_3
1	0	1
1	0	0
0	1	1
1	1	0

Set of DMLP examples Δ associated with r:

$$p_r((1,0,1)) \leftarrow$$
$$p_r((1,0,0)) \leftarrow$$
$$p_r((0,1,1)) \leftarrow$$
$$p_r((1,1,0)) \leftarrow$$

DMLP background theory Γ:

$$p_1(x) \leftarrow x[A_1] = 1$$
$$p_2(x) \leftarrow x[A_2] = 1, x[A_3] = 0$$
$$p_2(x) \leftarrow x[A_1] = 1$$

3.3. Semantics

3.3.1. Probability function.

DEFINITION 3.3.1. Let θ be a set of tuples $\theta = \{v_1/\sigma_1, \dots, v_n/\sigma_n\}$. θ is called *substitution* where each v_i is a variable and each σ_i is a tuple of terms and for no distinct i and j is v_i the same as v_j.

Let F be a formula, then $F\theta$ is called *instantiation* of F iff each occurrence of variable v_i in $F\theta$ is replaced by the tuple of ground terms σ_i $\forall i \in 1, \dots, n$ (i.e. $F\theta$ is the ground formula).

Let S_p be the definition of predicate p, then $S_p\theta$ is *instantiation* of the definition of the predicate p iff for each clause $C \in S_p$ holds that $C\theta$ is instantiation of formula C.

DEFINITION 3.3.2. Let r be a database, Γ be a DMLP background theory and Δ be the set of DMLP examples associated with r. Let F be a well-formed formula or a definition of some predicate symbol and $\overline{x}$ be a set of free variables in F ranging over tuples of r. By notation $F(r, \overline{x})$ we mean set of all tuples $a \in r$ for which exists instantiation θ such, that $F\theta$ is deductively satisfied in $\Gamma \cup \Delta$ (we write $\Gamma \cup \Delta \models F\theta$). If F is an empty formula then $F(r, \overline{x}) = r$.

In other words, $F(r, \overline{x}) = \{a \in r | \exists\theta\exists i : (x_i/a) \in \theta \wedge \Gamma \cup \Delta \models F\theta \wedge \overline{x} = (x_1, \dots, x_k)\}$. Therefore we write $F(r, \overline{x}) \subseteq r$, or say that $F(r, \overline{x})$ is the *selection* from r according to F. Because the definition of the predicate p can be seen as a disjunction of clauses, we can also say that DMLP clause, or the definition of the predicate (union of selections) p is the *selection* from the database r.

NOTE 3.3.3. Note that in the definition of instantiation of formula, we say that each variable is replaced by the tuple of ground terms and variables are syntactically constrained to range only over tuples of some database r. Simple corollary of this is that all predicates which say something about the given database r are of the same fixed arity as tuples of r. Also if we omit labels of DMLP clauses, we can see all predicates as definitions of selections from the database r, which defines partial ordering of selections and this allows us to compare two relations of such kind by usage of inclusion operator.

Now we will define the *probability* of clauses and definitions of predicate symbols. In fact it will be somehow a complementary term to *error measure* defined by Mannila in [**JMW 96**].

DEFINITION 3.3.4. Let A, B be well-formed formulas and $\overline{x_A}$, $\overline{x_B}$ sets of free variables used in them, where the set of variables $\overline{x_A}$ is distinct from the set $\overline{x_B}$. Let G be a function that assigns to every couple (r, C), where r is a database over the relational schema $R = (A_1, \dots, A_n)$ and $C = A \leftarrow B$ is a Horn clause, a value $q \in \mathbb{R} \cap [0, 1]$ such that:

(1) $G(r, C) = 0 \Longleftrightarrow A(r, \overline{x_A}) \cap B(r, \overline{x_B}) = \emptyset$
(2) $G(r, C) = 1 \Longleftrightarrow B(r, \overline{x_B}) \subseteq A(r, \overline{x_A})$

(3) Let also $C' = A' \leftarrow B'$ be a Horn clause. Then
$(\Gamma \cup \Delta \models (B \leftrightarrow B' \wedge A \rightarrow A')) \implies G(r, C) \leq G(r, C')$[1], where Γ is a DMLP background theory, and that is the same as
$B'(r, \overline{x_{B'}}) = B(r, \overline{x_B}) \wedge A'(r, \overline{x_{A'}}) \subseteq A(r, \overline{x_A}) \implies G(r, C) \leq G(r, C')$

Let G be a function from the class of functions defined above. We say that value $G(r, C)$ is a *probability of clause* C and G is a *probability function*.

NOTE 3.3.5. While $F(r, \overline{x}) = r$, where F is an empty formula and $\overline{x}$ is the set of free variables in F it comes out that if $C = A \leftarrow B$ and B is an empty formula, then $C(r, \overline{x_C}) = A(r, \overline{x_A})$, where $\overline{x_C}$ and $\overline{x_A}$ are corresponding sets of free variables. This works because each clause $C \in \Gamma$ contains formula in the form of $p_r(x)$ in its body where p_r is a predicate symbol associated with r. Therefore it is excluded that there exists a clause $C \in \Gamma$, which has an empty body.

Similar note holds when A is an empty formula. However, it is quite devious to define probability of a goal, while it would mean the probability of the question. Under the probability of the question we mean the question on the probability of the fact given as a goal. Therefore $G(r, \leftarrow B) = G(r, B)$. We will not use denotation $G(r, \leftarrow B)$, we will not even assign probability to goals.

In fact, the function G is able to evaluate every selection from some given database r. It is a measure of descriptive power of a selection according to r.

As it was mentioned above, the definition of probability function G is somehow complementary to the definition of an error measure in [**JMW 96**]. As Mannila claims, many functions will meet requirements on probability of selection. Many times also those which we consider bad for our purposes. However, we give an example of the probability function which is reasonable for our goals and we will use this function in the next parts of this work. Therefore it will be noted by symbol G, although it can be seen as an ambiguity. Still we allow some other definitions of the function G for other purposes.

DEFINITION 3.3.6. Let r be a non-empty database, C be a clause and $\overline{x_A}$, $\overline{x_B}$ corresponding sets of those free variables . By $|r|$ we mean the number of rows of the database r.

If C is holds the form of $C = A \leftarrow B$, where A and B are non empty formulas then:

$$G(r, C) = \begin{cases} \frac{|A \wedge B(r, \overline{x_A} \cup \overline{x_B})|}{|A(r, \overline{x_A})|} & \text{if } |A(r, \overline{x_A})| > 0 \\ 0 & \text{if } |A(r, \overline{x_A})| = 0 \end{cases}$$

If C is a clause $C = A \leftarrow$, where A is a non empty formula then:

$$G(r, C) = \begin{cases} \frac{|A(r, \overline{x_A})|}{|r|} & \text{if } |r| > 0 \\ 0 & \text{if } |r| = 0 \end{cases}$$

As it is obvious, the result of the given function G represents the conditional probability of $A \wedge B$ assuming that A holds, when clause given as an input has non empty head

[1]While given expression can be hard to read, please note that the operator of logical following $\models$ has higher priority than $\cup$, $\leftrightarrow$ and $\rightarrow$ and these have higher priority that $\wedge$, $\vee$ and finally $\neg$.

and body. For clauses with an empty body it returns probability of A according to r. As it was said in the note 3.3.5, we do not define probability function G for goals.

In fact, especially for clauses with an empty body, G returns the value which is similar to *confidence* of clause C in literature about data mining (e.g. [**JMW 96**]). Function G applied on DMLP clauses can also be interpreted as a function returning conditional probability under which some $a \in r$ satisfies the body of the clause assuming that it satisfies its head, or (for clauses with an empty body) probability under which the head of the clause holds for some $a \in r$.

NOTE 3.3.7. This definition allows us to speak about replacement of the head of the clause by its body and to compute the probability with which the resulting clause holds. This mechanism will be used in the next parts of this work, where we will define semantics of an evaluation function.

We can extend function G defined above in the definition 3.3.4 for selections in general.

DEFINITION 3.3.8. Let r be a database and $S_q = \{A_0 \leftarrow B_0, A_1 \leftarrow B_1, \ldots, A_n \leftarrow B_n\}$ be the definition of a predicate q. Let also $\overline{x_{A_0}}, \overline{x_{B_0}}, \ldots, \overline{x_{A_n}}, \overline{x_{B_n}}$ be sets of free variables corresponding to clauses from S_q. Let $\mathcal{A} = \bigcup_{i=0}^{n} A_i(r, \overline{x_{A_i}})$ and $\mathcal{B} = \bigcup_{i=0}^{n} B_i(r, \overline{x_{B_i}})$. We define the *probability of the selection* q by applying the definition of the function G on sets $\mathcal{A}$ and $\mathcal{B}$.

$$G(r, S_q) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A}|}$$

Now we are able to define the relation of logical following $\models_G$ according to the function G straightforwardly.

DEFINITION 3.3.9. Let $D = g{:}C$ be a DMLP clause, G be the probability function and r be a database over a relational schema R.

$$r \models_G D \iff G(r, C) \geq g$$

Let Φ be a DMLP program and let $D \in \Phi$. Now

$$r \models_G \Phi \iff \forall D \in \Phi : r \models_G D$$

NOTE 3.3.10. The previous definition says that the probability that the DMLP clause $D = g{:}C$ follows (we mean *following* in terms of the previous definition) from the database r is at least g.

3.3.2. Evaluation function. When we have already defined constraints and restrictions under which some DMLP program can be induced from the given database r, we will define model semantics for such DMLP programs and we will try to show that the database can be a probabilistic model of such induced program. But at first we will need some additional definitions which will help us with this approach.

DEFINITION 3.3.11. Let Φ be a database DMLP program, Γ be a DMLP background theory of Φ, Δ be a set of DMLP examples and $C = \leftarrow p(a)$ be a goal, where a is a tuple of ground terms and $p \in_{def} \Gamma$ is a predicate symbol. Function Ξ is defined as follows

$$\Xi(\Delta, \Gamma, \Phi, C) = q$$

where $q \in \mathbb{R} \cap [0, 1]$ is called *DMLP evaluation function.*

Again, by the definition above, we defined the whole class of evaluation functions. In general an evaluation function estimates the probability of any given goal according to DMLP examples, DMLP background theory, and induced DMLP program.

As it was said for the probability function G, many functions will meet requirements on an evaluation function. As we did for an example of probability function, we give an appropriate example on an evaluation function, which we will use in the rest of this work. On the ground of proof of DMLP program, we will define instance of an evaluation function which will be used later. It will work in the similar way as an SLD refutation. But at first let us define a few additional terms.

NOTE 3.3.12. In the next text we will use denotation $P(\Phi)$ for a logic program which contains clauses of DMLP program Φ with labels removed.

DEFINITION 3.3.13. Let Γ be a background theory and G the probability function. We say that Γ_{base} is a *DMLP background base* when $\Gamma_{base} = \{G(r, C){:}C | \forall C \in \Gamma\} \cup \{G(r, C_e){:}C_e \leftarrow | \forall C_e \in_e \Gamma\}$ where under C_e we understand all basic equalities contained in the clauses of Γ. Under the expression $C_e \in_e \Gamma$ we understand that there exists a clause $C \in \Gamma$ which contains the basic equality C_e in its body.

Γ_{base} contains the union of DMLP clauses labeled by the function G and the set of labeled basic equalities which can be found in Γ. As it is obvious, Γ_{base} is a DMLP program.

EXAMPLE 3.3.14. DMLP background base Γ_{base} constructed according to r, Δ and Γ from example 3.2.15 on page 22:

$$\begin{array}{rcl}
0.75 & : & x[A_1] = 1 \leftarrow \\
0.5 & : & x[A_2] = 1 \leftarrow \\
0.5 & : & x[A_3] = 0 \leftarrow \\
1.0 & : & p_1(x) \leftarrow x[A_1] = 1 \\
0.33 & : & p_2(x) \leftarrow x[A_2] = 1, x[A_3] = 0 \\
1.0 & : & p_2(x) \leftarrow x[A_1] = 1
\end{array}$$

DEFINITION 3.3.15. *DMSLD inference rule* takes a goal $g_1{:}C_1$ and a labeled clause $g_2{:}C_2$ and produces a labeled goal $(g_1 \cdot g_2){:}C_R$, where C_R is an SLD resolvent of C_1 and C_2. If there is no SLD resolvent for clauses C_1 and C_2, there is no DMSLD resolvent of them, too.

DEFINITION 3.3.16. Let Φ be a DMLP program, Γ_{base} be a DMLP background base and C be a goal. *DMSLD refutation* of the goal C is the repeated application of the DMSLD inference rule in the same manner as the SLD refutation does. The DMSLD refutation can be represented in the form of sequence of clauses, used to produce partial resolvents of the whole refutation, $\langle 1{:}C, g_1{:}C_1, \ldots, g_n{:}C_n\rangle$, where C is a goal and $\forall 1 \leq i \leq n : (g_i{:}C_i) \in \Phi \cup \Gamma_{base}$ and $\langle C, C_1, \ldots, C_n\rangle$ is a an SLD refutation of C from $P(\Phi \cup \Gamma_{base})$.

We are prepared to define an evaluation function Ξ_1 on this basis. The definition of it will be divided into three steps. In the definition 3.3.17 we will show how function Ξ_1 is able to estimate label of a simple goal with a constant, in the definition 3.3.21 we will extend it to the estimation of labels of goals with a variable in them. Finally in the definition 3.3.23 we will extend function into its full power. We will show how it estimates labels for whole clauses. As far as the definition of an estimation function Ξ_1 is quite complicated we give also few examples near definitions.

DEFINITION 3.3.17. Let r be a relational database over a relational schema $R = (A_1, A_2, \ldots, A_n)$, Γ be a DMLP background theory, Φ be a DMLP program and $C = \leftarrow p(a)$ be a goal where $p \in_{def} \Gamma$ and $a \in A_1 \times A_2 \times \cdots \times A_n$ is a tuple of terms. Let also $\gamma = \langle 1{:}C, g_1{:}C_1, \ldots, g_n{:}C_n\rangle$ be a sequence of DMLP clauses representing a DMSLD refutation of the goal C. There can exist more than one such refutation, or even none.

Answer probability of an evaluation function Ξ_1 is the value:

$$\Xi_1(\Delta, \Gamma, \Phi, C) = \begin{cases} \max_{\forall\gamma}(\prod_{i=1}^{n} g_i) & \text{if there is some DMSLD refutation } \gamma \\ 0 & \text{if there is no DMSLD refutation for } C \end{cases}$$

NOTE 3.3.18. The first member of the DMSLD refutation is the goal labeled by the value 1. According to the note 3.2.5, in the body of the clause we have only negative literals, which do not have to hold and therefore they have to have label 0. While DMSLD refutation is based on the proof by contradiction, where we start from negation of the goal, we are allowed to start the refutation from the goal $1{:}C$.

This had to be done, because according to the note 3.3.5, the probability of the DMLP goal is not defined.

EXAMPLE 3.3.19. DMSLD refutation and computation of Ξ_1 for goal with a constant constructed according to r, Δ and Γ from example 3.2.15 and Γ_{base} from example 3.3.14. For simplicity, we will use an empty DMLP program Φ. It will not influence the computation, while DMLP background base Γ_{base} is DMLP program too.

Let us have the goal $\leftarrow p_2((1, 1, 0))$. We can compute three DMSLD refutations for it. These are shown in the next table.

$\prod_{i=1}^{n}$	γ
0.0825	$0.33{:}p_2(x) \leftarrow x[A_2] = 1, x[A_3] = 0$; $0.5{:}x[A_3] = 0 \leftarrow$; $0.5{:}x[A_2] = 1 \leftarrow$
0.0825	$0.33{:}p_2(x) \leftarrow x[A_2] = 1, x[A_3] = 0$; $0.5{:}x[A_2] = 1 \leftarrow$; $0.5{:}x[A_3] = 0 \leftarrow$
0.75	$1.0{:}p_2(x) \leftarrow x[A_1] = 1$; $0.75{:}x[A_1] = 1 \leftarrow$

In the first column, there is a value of product of labels used in the given refutation. In the second column we show a chain of clauses used in each refutation. The final result of the function Ξ_1 for the goal $\leftarrow p_2((1,1,0))$ is

$$\Xi_1(\Delta, \Gamma, \emptyset, \leftarrow p_2((1,1,0))) = 0.75$$

Let us take a closer look into the definition of Γ_{base}. It consists of a set of labeled clauses constructed from a DMLP background theory Γ and a set of labeled basic equalities. While these basic equalities are clauses with an empty body, the probability function G treats them as if they have expression $p_r(x)$ in the body, where p_r is the predicate symbol associated with a database r and x is the variable contained in the basic equality. Because of this, the label of a basic equality can be seen as a proportion of a database which is covered by a selection defined by it. This is the property of all DMLP clauses with an empty body.

Labels of clauses with non-empty body can be seen as a proportion of a part of a database, where the head and the body hold in conjunction, to the size of the part of the database in which only body of the clause holds. In other words, if we had two selections, one defined as a part of the database where the body of the given clause holds and the other as a part where the head and the body of the given clause hold together, label of such clause would be the size of the second selection divided by the size of the part of the database defined by the first selection. This is exactly what function G does.

Here we will care only about clauses with an empty body. One of good methods to estimate proportion of conjunction of basic equalities to the size of the whole database would be a minimalistic approach. By this we mean calculation of minimal possible intersection of segments of the database. Although we know the proportion of the database in which the given member of the conjunction holds, we do not know exactly which part it is. We only know its size. Of course, size of such intersection depends on particular parts of the database (we need to know whether given row lies in the intersection of members of the conjunction or not). Because we do not know this information, we can only guess the size of such an intersection. From sizes of parts of the database where the given expressions hold we can estimate the size of the least intersection where the conjunction of them must hold (this can easily be zero). Again we do not know which part it is. We just know its size.

NOTE 3.3.20. We can estimate the label of the least intersection of two labeled selections $g_1{:}S_1$ and $g_2{:}S_2$ as $\max(g_1 + g_2 - 1, 0){:}(S_1, S_2)$. This equation can be extended to tuple of selections. Let us have selections $g_1{:}S_1, \ldots, g_n{:}S_n$. Probability of their least intersection can be estimated as

$$\Xi_1(\Delta, \Gamma, \Phi, S_1 \wedge \cdots \wedge S_n) = \max((\sum_{i=1}^{n} g_i) - (n-1), 0)$$

It is easy to see that $\forall i : \Xi_1(\Delta, \Gamma, \Phi, S_1 \wedge \cdots \wedge S_n) \leq g_i$.

DEFINITION 3.3.21. Let us have similar assumptions as in the definition 3.3.17. Let $C = \leftarrow p(x)$ be a goal with a variable x and a predicate symbol $p \in_{def} \Gamma$.

For each DMSLD refutation $\gamma = \langle 1{:}C, g_1{:}C_1, \ldots, g_n{:}C_n\rangle$ we can construct a set of basic equalities $\gamma_e = \{g_k{:}C_k | \exists k : 1 \leq k \leq n \wedge g_k{:}C_k \in \gamma\}^2$, where each C_k is a basic equality and the set consisting of the rest of the given DMSLD refutation $\gamma_{\overline{e}} = \{g_l{:}C_l | \exists l : 1 \leq l \leq n \wedge g_l{:}C_l \in \gamma\}$, where each C_k is a clause with non empty head and body. Conjunction of these basic equalities can be seen as a selection defining predicate p. Under

$$\omega_{\gamma_e} = \max((\sum_{\forall g_i:C_i \in \gamma_e} g_i) - (|\gamma_e| - 1), 0)$$

we understand proportion of the least intersection of selections defined by the basic equalities from γ_e.

Answer probability of function Ξ_1 for such a goal will be

$$\Xi_1(\Delta, \Gamma, \Phi, C) = \begin{cases} \max_{\forall \gamma}((\prod_{g_i:C_i \in \gamma_{\overline{e}}} g_i) \cdot \omega_{\gamma_e}) & (1) \\ 0 & (2) \end{cases}$$

The first equation holds when there exists some DMSLD refutation of the goal C. Answer probability of Ξ_1 is 0, when there is no DMSLD refutation of the given goal C. Note that we used equation from the note 3.3.20 to compute the estimation of the least intersection of basic equalities.

EXAMPLE 3.3.22. DMSLD refutation and computation of Ξ_1 for a goal with a variable constructed according to r, Δ, and Γ, from the example 3.2.15 and Γ_{base} from the example 3.3.14. For simplicity, we will use again an empty DMLP program Φ. It will not influence the computation, because DMLP background base Γ_{base} is a DMLP program too. Whole DMSLD refutations can be found in the example 3.3.19

Let us have the goal $\leftarrow p_2(x)$. We can compute three DMSLD refutations for it. These are shown in the next table.

$\prod$	ω_{γ_e}	γ_e	$\gamma_{\overline{e}}$
0.0	0.0	$0.5{:}x[A_3] = 0 \leftarrow;\ 0.5{:}x[A_2] = 1 \leftarrow$	$0.33{:}p_2(x) \leftarrow x[A_2] = 1, x[A_3]$
0.0	0.0	$0.5{:}x[A_2] = 1 \leftarrow;\ 0.5{:}x[A_3] = 0 \leftarrow$	$0.33{:}p_2(x) \leftarrow x[A_2] = 1, x[A_3]$
0.75	0.75	$0.75{:}x[A_1] = 1 \leftarrow$	$1.0{:}p_2(x) \leftarrow x[A_1] = 1$

In the last two columns the table above shows parts of DMSLD refutations divided into two groups γ_e and $\gamma_{\overline{e}}$. The second column shows the value of $\omega_{\gamma_e}$and finally in the first column there is the value of the product $(\prod_{g_i:C_i \in \gamma_{\overline{e}}} g_i) \cdot \omega_{\gamma_e}$

$$\Xi_1(\Delta, \Gamma, \emptyset, \leftarrow p_2(x)) = 0.75$$

Finally we will extend the function Ξ to be able to compute also labels of the whole clauses.

DEFINITION 3.3.23. Let us have the same assumptions as in the definitions3.3.17 and 3.3.21. Let $C = A \leftarrow B_1, \cdots, B_n$ be a range restricted definite clause with a variable x

[2]Although γ is not the set, we used expression $g_k{:}C_k \in \gamma$. By that we mean that the given DMLP clause is one of the members of γ.

and a predicate symbol $p \in_{def} \Gamma$. Let us also have return values of the function Ξ_1 for the goals $\leftarrow A, \leftarrow B_1, \cdots, \leftarrow B_n$. We can compute the estimation of the label on the right side of the clause by using the equation from the note 3.3.20 as

$$\Xi_1(\Delta, \Gamma, \Phi, B_1 \wedge \cdots \wedge B_n) = \max((\sum_{i=1}^{n} \Xi_1(\Delta, \Gamma, \Phi, \leftarrow B_i)) - (n-1), 0)$$

Answer probability of the function Ξ_1 for such clause is

$$\Xi_1(\Delta, \Gamma, \Phi, C) = \Xi_1(\Delta, \Gamma, \Phi, \leftarrow A) \cdot \Xi_1(\Delta, \Gamma, \Phi, B_1 \wedge \cdots \wedge B_n)$$

Just like after the definition of the probability function G, we decided to use general notation G for the particular probability function. In the rest of this text we will use notation Ξ for function Ξ_1, because it suits our needs.

Because the definition of the evaluation function Ξ may seem to be quite complicated and foggy, we have to explain it. For simple goals, which are the goals with one predicate symbol without any variables, the function Ξ evaluates all possible deductive proofs of the given ground atom and chooses the one which assigns the maximal conditional probability to the goal. This is the conditional probability that if the given ground atom belongs to the given database r it holds in it as well, according to the given DMLP background base and the DMLP program. Note that Ξ evaluates only those DMSLD refutations for the given goal, for which all basic equalities appearing in the refutation hold.

For more complicated goals, which are those including a predicate with a variable, it estimates the maximal probability that the given clause holds in the database r according to alternative definitions of the goal (i.e. conjunction definitions of predicates in the goal) in DMLP background theory Γ. Because the only constructions, which define selections, are basic equalities in DMLP background theory Γ, the definition of each member of the goal consisting only of basic equalities must be built from them. This is done for each possible proof (DMSLD refutation) of the given goal. At this point we can divide the refutation into two parts. One is constructed by using clauses which only substitute members of the resolvent by other predicates, which have to be substituted in the next steps of refutation, and the second part, consisting of basic equalities appearing in the definition of the selection which is finally represented by the given goal. By computing the product of labels of the first part of the refutation we compute the probability that the goal can be represented by the conjunction of computed basic equalities and for the second part we have to compute the proportion of the least part of the database r which possibly can be described by these basic equalities. We simply compute the least intersection of selections defined by them. By multiplication of these two numbers we have the size of the minimal proportion of the database r which can be described by given goal. In other words, we have the probability with which some tuple of atoms belongs to the database r if it satisfies the given goal.

Finally, for the whole clauses, the function Ξ estimates the value with which both the body and the head of the clause hold in the database. This value represents an estimation of the probability with which we can replace the head of the clause by its body. Although this is a bold statement, we will prove it in the theorem 3.3.26.

Note that the function Ξ estimates the least probability that atom holds without looking into the database. By this we mean, that an algorithm for estimation of probability of a goal, described in the definition of function Ξ, does not need to use the database to compute the probability of a given goal.

In the rest of this part of the text we will show what the relation between Ξ and G is. At first we will prove lemma which will be used in the proof of the theorem 3.3.26.

LEMMA 3.3.24. *Let Γ_{base} be a DMLP background base constructed to some r, Δ and Γ. Let also Φ be a DMLP program. Let finally $\leftarrow p(a)$ be a goal with a constant a. Then*

$$\Xi(\Delta, \Gamma, \Phi, \leftarrow p(a)) \leq G(r, p(a))$$

PROOF. We will use induction to prove the lemma above. We will show that for every refutation of the goal $\leftarrow p(a)$ value of the probability function G, $G(r, p(a))$ is higher than an estimation by Ξ.

(1) If there is no refutation of $\leftarrow p(a)$, then $\Xi(\Delta, \Gamma, \Phi, \leftarrow p(a)) = 0$. From this simply follows that the lemma 3.3.24 holds.
(2) Let us assume that there is no clause, with the predicate symbol p in the head, in Φ. Then any refutation γ of the goal $\leftarrow p(a)$ must contain a clause from Γ_{base} with the predicate p in its head. In fact it will be the first member of the refutation γ. From the definition 3.3.21 we see that the final result of the evaluation function Ξ is a product of labels of members (we mean members which are clauses with non-empty head and body) of the refutation. Therefore $\exists g_k{:}C_k \in \Gamma_{base}$ which is the member of γ. From that we have that $\prod_{i=1}^{n} g_i \leq g_k$. Because for each $g_j{:}C_j \in \Gamma_{base}$ we have that $g_j = G(r, C_j)$, we can say that $\forall\gamma\exists k \prod_{i=1}^{n} g_i \leq G(r, C_k)$ what completes the proof in this step. (Although the probability function is not defined for the goal $C = \leftarrow p(x)$, for simplicity we use denotation $G(r, C)$ for the expression $G(r, p(x))$).
(3) Now assume that there is already one clause with the predicate symbol p in its head in Φ. Then there is a refutation γ which was chosen because it leads to the maximal return value of Ξ. But this refutation must contain the clause $g_k{:}C_k$ from either Γ_{base} or Φ which contains the predicate p in its head.
    (a) if $g_k{:}C_k$ is from Γ_{base} then the step 2 applies.
    (b) if $g_k{:}C_k$ is from Φ then $g_k \leq g_l$, where g_l is the label of a clause from Γ_{base}, because according to the step 2, when computing the label for $g_k{:}C_k$ some clause $g_l{:}C_l \in \Gamma_{base}$ had to be used.
(4) If there is more than one clause with the predicate p in the head, when we are computing the label for the goal $\leftarrow p(a)$, then by recursive application of the rules 2 and 3 we can state that lemma holds.

□

NOTE 3.3.25. Lemma 3.3.24 is applicable also to goals with a variable, but the proof would be a bit complicated because of many math symbols in it. However, its skeleton would be the same. Therefore we gave only simplier version of it.

THEOREM 3.3.26. *Let Γ_{base} be a DMLP background base constructed according to some r, Δ and Γ. Let also Φ be a DMLP program and $C = A \leftarrow B$ be a clause with non-empty head and body. Then*

$$\Xi(\Delta, \Gamma, \Phi, C) \leq G(r, C)$$

PROOF. From the definition 3.3.23 we have that $\Xi(\Delta, \Gamma, \Phi, C) = \Xi(\Delta, \Gamma, \Phi, \leftarrow A) \cdot \Xi(\Delta, \Gamma, \Phi, B_1 \wedge \cdots \wedge B_n)$. From the lemma 3.3.24 we have that $\Xi(\Delta, \Gamma, \Phi, \leftarrow A) \leq G(r, A)$ and $\forall i : \Xi(\Delta, \Gamma, \Phi, \leftarrow B_i) \leq G(r, B_i)$. Finally from the note 3.3.20 we can see that $\forall i : \Xi(\Delta\,\Gamma\,\Phi\,B_1 \wedge \cdots \wedge B_n) \leq \Xi(\Delta, \Gamma, \Phi, B_i)$. It is also easy to see that $\forall A : \frac{|A(r,\overline{x_A})|}{|r|} \leq 1$.

After this preparation the proof follows:

$$\begin{array}{ll}
\frac{|A(r,\overline{x_A})|}{|r|} \leq 1 \leq \frac{1}{\frac{|A(r,\overline{x_A})|}{|r|}} & / \cdot \frac{|A \wedge B(r,\overline{x_A} \cup \overline{x_B})|}{|r|} \\
\frac{|A(r,\overline{x_A})|}{|r|} \cdot \frac{|A \wedge B(r,\overline{x_A} \cup \overline{x_B})|}{|r|} \leq \frac{\frac{|A \wedge B(r,\overline{x_A} \cup \overline{x_B})|}{|r|}}{\frac{|A(r,\overline{x_A})|}{|r|}} & \\
\frac{|A(r,\overline{x_A})|}{|r|} \cdot \frac{|A \wedge B(r,\overline{x_A} \cup \overline{x_B})|}{|r|} \leq \frac{|A \wedge B(r,\overline{x_A} \cup \overline{x_B})|}{|A(r,\overline{x_A})|} &
\end{array}$$

From the definition 3.3.4 of the probability function and preparations at the beginning of the proof, we finally conclude:

$$\Xi(\Delta, \Gamma, \Phi, \leftarrow A) \cdot \Xi(\Delta, \Gamma, \Phi, A \wedge B) \leq G(r, A) \cdot G(r, A \wedge B) \leq G(r, A \leftarrow B)$$

This completes the proof. □

3.3.3. Semantics of DMLP. In this subsection we define distributional L-interpretation and distributional L-model in the similar way as Muggleton does in [**Mug 00**] for SLPs.

DEFINITION 3.3.27. Let L be a first-order language and D_p is a probability distribution over ground atoms of the predicate symbol p in L. If I is a vector consisting of one such distribution for every predicate symbol $p \in L$, then I is called a *distributional L-interpretation* (or simply *interpretation*).

If $A \in L$ is an atom with the predicate symbol p and I is an interpretation, then $I(A)$ is the probability of A according to D_p in I.

Let Φ be a DMLP program. Let L_Φ be the smallest first-order language involving at least one constant and a predicate symbol of Horn theory $P(\Phi)$. In this case a distributional L_Φ-interpretation is called a *distributional Herbrand interpretation* of Φ.

Now we are finally prepared to define the distributional model of DMLP program.

DEFINITION 3.3.28. We say that an interpretation M is a *distributional L-model* (or simply *model*) of the given DMLP Φ with a background theory Γ and a set of DMLP examples Δ, iff

$$\forall a \in r : M(p(a)) \geq \Xi(\Delta, \Gamma, \Phi, p(a))$$

where p is a predicate from Γ and a is a tuple of terms from a database r.

Let us take a closer look at the distributional L-interpretation defined by the probability function G. At first sight our intuition says that it will define the real probability of atoms which hold in the database r. In the next definition we will define such interpretation.

DEFINITION 3.3.29. Let Δ be a set of DMLP examples associated with a relational database r and Γ be a DMLP background theory. We say that I_G is a *natural database interpretation* iff I_G is defined as follows:

$$\forall p \in_{def} \Gamma, \forall a \in r : I_G(p(a)) = G(r, p(a)) = 1$$

Although the definition of such artificial interpretation, as I_G surely is, seems to be quite strange, it will be helpful in the next chapter, where we will define operators of DMLP induction.

THEOREM 3.3.30. *Let Δ be a set of DMLP examples associated with a relational database r and Γ be a DMLP background theory associated with a DMLP program Φ. If a interpretation I_G is the natural database interpretation of DMLP program Φ it is also a distributional L-model of it.*

PROOF. From the definition of Ξ we have that for every Δ, Γ, Φ and a goal C, $\Xi(\Delta, \Gamma, \Phi, C) \leq 1$. Requirements on the interpretation I_G restrict only probability values of atoms containing tuples a, which belong to database r. Therefore we can say that $\forall p \in_{def} \Gamma, \forall a \in r : 1 = I_G(p(a)) \geq \Xi(\Delta, \Gamma, \Phi, p(a)) \geq 0$. From that we can easily see that I_G is a distributional L-model of any Φ induced from the given database r and the DMLP background theory Γ. $\square$

NOTE 3.3.31. According to the theorem 3.3.30, we can say that any natural database interpretation I_G is a *natural model* of a database r.

DEFINITION 3.3.32. Let L be a language. Let $\epsilon \in \mathbb{R} \cap [0,1]$ be a real value. L-interpretation M is called an *ϵ-tight distributional L-model* (or simply an *ϵ-model*) iff

$$\forall A \in L : 0 \leq I_G(A) - M(A) \leq \epsilon$$

where A is a ground atom containing only tuples of terms a belonging to the given database r. The real value ϵ will be called *tightness* of the model M.

If M is an ϵ-model of Φ and M is Herbrand interpretation with respect to Φ, then M is an *ϵ-tight distributional Herbrand model* of Φ (or simply *Herbrand ϵ-model*).

DEFINITION 3.3.33. Let r be a database, Γ be a DMLP background theory of a DMLP program Φ. We say that distributional interpretation I_Ξ is an *estimated interpretation of* Φ iff

$$\forall p \in_{def} \Gamma, \forall a \in r : I_\Xi(p(a)) = \Xi(\Delta, \Gamma, \Phi, p(a))$$

NOTE 3.3.34. From the definition of distributional L-model and the estimated interpretation of a DMLP program Φ we simply have that I_Ξ is also a distributional L-model of Φ. Therefore we can say that I_Ξ is an *estimated model of* Φ.

DEFINITION 3.3.35. Let r be a database and Γ be a DMLP background theory. Suppose that Φ_1 and Φ_2 are DMLP programs constructed from r and Γ. We will write that $\Phi_1 \models_{\Xi} \Phi_2$ iff every distributional L-model of Φ_1, is also a distributional L-model of Φ_2.

As we did in the definition of $\models_{\Xi}$, we will write that $\Phi_1 \models_{\Xi,\epsilon} \Phi_2$ iff every ϵ-model of Φ_2 is an ϵ-model of Φ_1, too.

CHAPTER 4

Algorithms of DMLP induction

In this section we will propose and discuss algorithms and approaches to the DMLP induction. At first we will speak more formally about a problem of a DMLP induction and then we will propose different operators of DMLP induction.

4.1. Operators of DMLP induction

Now we are finally prepared to define the DMLP induction problem, what is the core of this work.

DEFINITION 4.1.1. Let r be a database, Φ be a DMLP program, Γ be a DMLP background theory, Ξ be the evaluation function, G be the probability function and $\epsilon \in \mathbb{R} \cap [0, 1]$ be a real value. Let also I_G be a natural model of r and I_Ξ be an estimated model of Φ.

Operator $Ind_{DMLP}(r, \Gamma, \Xi, G, \epsilon)$ is called *operator of DMLP induction* iff

(1) $Ind_{DMLP}(r, \Gamma, \Xi, G, \epsilon) = \Phi$
(2) $r \models_G \Phi$
(3) I_Ξ is an ϵ-tight distributional L-model of Φ (i.e. $\forall p \in_{def} \Gamma, \forall a \in r : 0 \leq I_G(p(a)) - I_\Xi(p(a)) \leq \epsilon$).

Search for appropriate DMLP program Φ by using some DMLP induction operator Ind_{DMLP} which will minimize the value of $I_G(p(a)) - I_\Xi(p(a))$ for all $a \in r$ and $p \in_{def} \Gamma$ will be called the *DMLP induction problem*.

NOTE 4.1.2. Point 2 in the definition 4.1.1 is the most important component of this definition, while it defines a characteristics of an induced DMLP program. Point 3 just strengths this requirement.

Verifying, whether the given DMLP program Φ in combination with the given DMLP induction operator Ind_{DMLP} leads to an ϵ-tight estimated model of Φ, can be quite computionally time-consuming task when we are working with a large database r and a large DMLP background theory Γ (because of the enumeration of all atoms in the form of $p(a)$, for all $p \in_{def} \Gamma$ and $a \in r$). From this reason we will also define weak DMLP induction problem.

DEFINITION 4.1.3. Let r be a database, Φ be a DMLP program, Γ be a DMLP background theory, Ξ be the evaluation function and finally let G be the probability function.

Operator $Ind^{weak}_{DMLP}(r, \Gamma, \Xi, G, \epsilon)$ is called *operator of weak DMLP induction* iff

(1) $Ind^{weak}_{DMLP}(r, \Gamma, \Xi, G, \epsilon) = \Phi$
(2) $r \models_G \Phi$

(3) Γ is the DMLP background theory of Φ

Search for an appropriate DMLP program Φ by using some weak DMLP induction operator Ind^{weak}_{DMLP} will be called the *weak DMLP induction problem*.

NOTE 4.1.4. Point 3 in the definition 4.1.3 is unimportant, however it says us that induced DMLP program Φ cannot be just any DMLP program, but that it must be DMLP program which uses only predicates defined in the DMLP background theory Γ and that it describes database r (because it uses DMLP background base Γ_{base} constructed from the given set of DMLP examples Δ associated with r).

4.1.1. Relation between G, Ξ, Ind_{DMLP} and Ind^{weak}_{DMLP}. As it was sketched in the text above a DMLP program Φ, which is the final product of a DMLP induction, can be seen as some kind of a description of the given database r. The function G is able to accurately label clauses of any DMLP program according to the database r. However, this can be quite time-consuming operation and in some applications of the DMLP induction problem we do not need an accurate probability of a given formula. An estimation of the probability of a formula can be sufficient for purposes of such application. For this estimation we can use the evaluation function Ξ, which is also able to label any formula, but there is a suspicion that Ξ will do that faster than G. Algorithm of Ξ also does not need to walk through the whole database r.

These facts are used by a DMLP induction operator Ind_{DMLP}. Although an operator is rather more abstract notion, here we see it as an algorithm of construction of DMLP program , model of which, estimated by an evaluation function Ξ, is as close as possible to an ideal model, computed by the probability function G. Still we do not restrict a DMLP induction operator to be only such algorithm.

As it is obvious, in some cases it may not be possible to induce any non-trivial DMLP program Φ from the given r and Γ. The quality of the induced DMLP program is represented by the tightness of an estimated model.

Although the definition of Ind^{weak}_{DMLP} can seem to be quite meaningless, such operator of the DMLP induction can be usefull in cases, where an application does not depend on the quality of the induced DMLP program. In the next text we will show applications where usage of such induction operator is appropriate.

4.1.2. General design of operator of the DMLP induction. At the first sight we could divide our approaches into two groups. Those based on precise computation from the database and the group of estimative strategies. This straightly offers us two algorithm designs according to the ideas described in the section 3.3. We can base a precise computing approach on the usage of the probability function G and an estimative approach on the evaluation function Ξ. In next two subsections we will present our ideas for development of an efficient algorithm of the DMLP induction.

If we start a top-down analysis of an operator of the DMLP induction, we can see it as an entity consisting of two modules. The first module will generate candidates for new rules and the second one will label them and decide whether the resulting DMLP clause satisfies constraints imposed on an output of such operator.

Both proposed algorithms will use similar skeleton for proposing most suitable candidates for induced rules based on the search for frequently occuring patterns in the database and for inducing final DMLP clauses from these frequent patterns. For this purpose we will adapt generic data mining algorithm for finding all frequent sets, described in the literature (i.e. [**Man 97b**]). At first we will describe the module for proposing candidates for induced clauses and then we will discuss the second module for finalizing the work.

NOTE 4.1.5. While the conjunction $\wedge$ is a commutative operation, in the next text we will treat all conjunctions as sets. Therefore we will use expressions like $C \setminus \{A\}$ for simplicity.

Let us have a class $\mathcal{P}$ of patterns or sentences that describe properties of the given data. Under the *pattern* we mean any sentence, an expression or a statement which says something about properties of the given data set. In our case we will use the syntax of DMLP clauses to encode such patterns. The task can be described as a computation whether the given pattern (clause) $p \in \mathcal{P}$ is frequent enough to produce an interresting rule. As Mannila claims in [**Man 97b**], generic data mining task is to find the set

$$PI(r, \mathcal{P}) = \{p \in \mathcal{P} | p \text{ occurs sufficiently often in } r \text{ and } p \text{ is interesting}\},$$

where PI is a set of all interesting and frequent patterns and r is a database.

We will adapt this formalism for our purposes. Let r be a relational database and Γ be a DMLP background theory. As a class of all patterns describing properties of the database r according to the DMLP background theory Γ, we will use the set Γ^* of all expressions, in the form of conjunction, which we can produce from expressions of the form $p(x)$, where $p \in_{def} \Gamma$ and x is a variable ranging over tuples of database r. Let us assume, that there exists a binary relation $\prec$, which defines partial ordering over the expressions of Γ^*. We will view our task as the problem of finding the sentences in Γ^* that are “sufficiently true” in the data. At this step we will not care about other aspects of induced rules, which can make them interesting enough to manipulate them later.

The algorithm 3 is based on the claim that if we find a frequent expressions, all of its subexpressions must be frequent and vice versa.

In the algorithm 3, we use three sets of expressions. $Tested$ is the set of all expressions which we have already considered, $Candidates$ is the set of possible candidates for frequent patterns in the next step and finally $FreqPatt$ is the set of frequent conjunctions, already found. In this algorithm we start from atomic expressions, which are the heads of selections defined in Γ. The relation $\prec$ is the relation of generalisation. As it is obvious, $(A \prec B) \iff (\forall C \in B \Rightarrow C \in A)$, where A and B are conjunctions of atomic expressions $C = p(x) \in \Gamma^*$ in the first step of the algorithm 3 (i.e. C stands for a member of such conjunction).

The only undefined symbol is the function $Estimate_{conj}(r, A)$. This is a function of estimation of the *frequency* of the given conjunction A. This is important difference

Algorithm 3 DMLP FFC - Finding all frequent conjunctions (adaptation of FFP algorithm from [**Man 97b**]).

```
$Tested = \emptyset$
$FreqPatt = \emptyset$;
$Candidates = \{p(x) | p \in_{def} \Gamma\}$;
while $Candidates \neq \emptyset$ do
  for each $A \in Candidates$ do
    if $Estimate_{conj}(r, A) > treshold$ then
      $FreqPatt = FreqPatt \cup \{A\}$
    else
      $Tested = Tested \cup \{C \in \Gamma^* | C \prec A\}$;
    end if
    $Candidates = Candidates \setminus \{A\}$
    $Candidates = Candidates \cup \{C \in \Gamma^* | C \prec A \wedge C \notin Tested\}$;
    $Tested = Tested \cup \{A\}$;
  end for
end while
```

between our two proposals for an DMLP induction operator. We will discuss this function more in subsections 4.1.3 and 4.1.4.

The algorithm 3 works simply. If it, during the computation, finds an expression which is frequent enough (i.e. its estimation of the frequency by the function $Estimate_{conj}$ is high enough), it adds all the expressions, which are more special than the expression found and it is subexpression of each of them, to the set of possible candidates for frequent conjunctions. If the actually tested expression is not frequent enough, we have to exclude all more specialised expressions from the future testing, because these have no more chance to be frequent.

Now we are prepared to give a description of the second module which computes clauses induced by an operator of DMLP induction.

Algorithm 4 DMLP CIR - computation of induced rules

```
$\Phi = \emptyset$
compute the set $FreqPatt$ of all frequent conjunctions;
for each $C \in FreqPatt$ do
  for each member $A$ of the conjunction $C$ do
    $C_A = (A \leftarrow C \setminus \{A\})$
    if $Estimate_{rule}(r, C_A) > treshold$ then
      $\Phi = \Phi \cup \{Estimate_{rule}(r, C_A){:}C_A\}$;
    end if
  end for
end for
```

We can see that the algorithm 4 is quite simple. It tries to create all possible range restricted Horn clauses from all frequent conjunctions and computes an estimation of the probability of such clauses. If the label is sufficiently high it adds such a clause to the induced DMLP program Φ. The only undefined symbol, alike above in the algorithm 3, is an estimation function $Estimate_{rule}$. The implementation of this function is again the main difference between our two proposals which we will discuss in the subsections 4.1.3 and 4.1.4.

As Mannila in [**Man 97b**] claims, it is possible to modify the implementation of this generic algorithm by usage of hill-climbing searches for the best conjunctions in the module implemented by the algorithm 3. As it is obvious, the architecture of our generic algorithm is very similar to the generic data mining system architecture. These facts will be discussed later in the section 5.

Customization of the generic algorithms 3 and 4 lies in the customization of the first estimation function $Estimate_{conj}(r, A)$, which estimates the frequency of the given conjunction A, and the second estimation function $Estimate_{rule}(r, C)$, which estimates the probability of the given clause (or the rule) C. By implementing these two functions in different ways we achieve two different algorithms with different complexity properties. Please note that we use the definition of weak DMLP induction operator (we use the definition of weak induction operator defined in the definition 4.1.3).

NOTE 4.1.6. Generic algorithm constructed from algorithms 4 and 3 with functions $Estimate_{rule}$ and $Estimate_{conj}$ unimplemented will be called the *generic skeleton* of a DMLP induction operator.

4.1.3. Precise approach using the probability function G. In this subsection we will define weak DMLP induction operator $Ind_G(r, \Gamma, G, \epsilon)$.

DEFINITION 4.1.7. Let r be a relational database, Γ be a DMLP background theory, G be the probability function and finally $\epsilon \in \mathbb{R} \cap [0, 1]$ be a treshold value. Let A be a conjunction of predicates defined in Γ and C be a range restricted definite clause constructed from predicates defined in Γ.

We say that a DMLP induction operator $Ind_G(r, \Gamma, G, \epsilon)$, which uses the generic skeleton, is *precise* when it implements estimation functions as follows

$$Estimate_{conj}(r, A) = G(r, A)$$

$$Estimate_{rule}(r, C) = G(r, C)$$

THEOREM 4.1.8. *Let Φ be a DMLP program induced by the precise operator Ind_G and Γ be its DMLP background theory. Then the operator Ind_G is the weak DMLP induction operator.*

PROOF. Proof of this theorem is quite straightforward. We only need to check whether each rule induced by the operator Ind_G satisfies requirements imposed on it by the definition of the weak DMLP induction operator.

The first and the third point of the definition are satisfied from the definition of operator Ind_G. We only need to check the satisfaction of the second point of the definition 4.1.3 (i.e. whether $r \models_G \Phi$). From the definition 3.3.9 we have that $r \models_G C \iff G(r, C) \geq g$, where $g{:}C \in \Phi$ is a DMLP clause. But from the definition 4.1.7 of operator Ind_G we have that this operator assigns to each clause label $g = G(r, C)$, therefore $g = G(r, C) \implies r \models_G C$. This completes the proof. $\square$

It is possible to implement the precise weak DMLP induction operator straightforwardly by usage of the probability function G, which we can implement as a query to the database r if it would be possible. Under additional strong restrictions on the syntax of a DMLP program we are able to express these queries in SQL. This will be discussed later.

NOTE 4.1.9. The computation of such query can lead to the recursive computation, therefore we have to additionaly constrain the form of clauses used as an input, or we can restrict the form of the DMLP background theory Γ. This note applies to the proposal in the section 4.1.4, too.

Induction operator as we defined it in the definition 4.1.7 is as precise as possible according to the given probability function G. It uses the computation from the database to label an input clause. As Mannila claims in [**Man 97b**], naive implementations of such operators can lead to slow operations for large databases. However, outputs of this approach are precise and terminal. There is nothing to improve on them.

4.1.4. Estimative approach using the evaluation function Ξ. The Weak DMLP induction operator $Ind_{\Xi}(r, \Gamma, \Xi, \epsilon)$, which uses the evaluation function Ξ, represents an estimative approach to a DMLP induction operator.

DEFINITION 4.1.10. Let r be a relational database, Δ be the set of DMLP examples associated with r, Γ be a DMLP background theory of a DMLP program Φ. Let also Ξ be the evaluation function and A be a conjunction of predicates defined in Γ and $C = C_{head} \leftarrow C_{body}$ be a range restricted definite clause constructed from predicates defined in Γ.

We say that a DMLP induction operator $Ind_{\Xi}(r, \Gamma, G, \epsilon)$, which uses the generic skeleton, is *estimative* when it implements the estimation functions as follows

$$Estimate_{conj}(r, A) = \Xi(\Delta, \Gamma, \Phi, A)$$

$$Estimate_{rule}(r, C) = \Xi(\Delta, \Gamma, \Phi, C)$$

THEOREM 4.1.11. *Let r be a relational database, Φ be a DMLP program, Γ be its DMLP background theory, Δ be the set of DMLP examples associated with r and finally Ξ be the DMLP evaluation function. Then the estimative operator Ind_{Ξ} is the weak DMLP induction operator.*

PROOF. Again we only need to prove that $r \models_G \Phi$ if we label clauses in Φ by usage of the estimative function Ξ. Proof of this straightforwardly follows from the proof of the theorem 3.3.26 on the page 32. $\square$

As it was already sketched in the subsection 4.1.1, the definition of the estimative induction operator Ind_{Ξ} is the main result of this chapter. In the theorem 4.1.11 we proved that the simple induction operator which uses the estimative function Ξ as the core of its computation is correct according to the definition of the simple induction operator. We designed this operator with the aim to produce an efficient method for an induction of DMLP programs. The estimative induction operator is not as precise as precise induction operators are, but we hope that under tight constraints on the syntax of DMLP programs they can produce quite good DMLP programs.

Finally let us discuss the complexity of induction operators proposed in the subsections 4.1.3 and 4.1.4.

4.2. Complexity of Ind_G and Ind_Ξ

For the proper specification of a complexity of any of given induction operators, we need to look into the generic algorithms of DMLP induction. These are the algorithm FFC (3) for finding all frequent conjunctions, and the algorithm CIR (4) for the final computation of induced rules from these frequent conjunctions.

Both algorithms, FFC as well as CIR use functions $Estimate_{conj}$ and $Estimate_{rule}$. Therefore we will at first analyze these functions.

In the case of the precise DMLP induction operator Ind_G both of these functions use the probability function G to compute a probability of the given clause or the conjunction. If we treat the function G as an algorithm to compute this probability, we will find that it needs to walk through the whole database r to find out for how many tuples, or rows, the given expression holds. However, before this verification it needs to compute an SLD refutation of the given expression in DMLP background theory Γ, what is some usuall logic program, to find all basic equalities which it needs to verify in the database. We need to compute an SLD refutation only from Γ because in the DMLP background theory, there is the only precise definition of each predicate used in any clause or a conjunction. While the complexity of the computation of such SLD refutation strongly depends on the syntactic constraints on Γ, we will not even estimate it. We just have to note, that in the case, when the given DMLP background theory Γ will contain recursive clauses, or when it allows recursive computation of the given logic program, it even may not be possible to compute any SLD refutation. Therefore we strongly suggest to impose as strong syntacic constraints on it, as it is possible. For example, it is well-advised to support only hierarchic logic programs[1] as DMLP background theory. It seems to suit all "normal" requirements on Γ. In this case, the computation of an SLD refutation from Γ can be linear according to the number of predicates defined in Γ. We will denote it by the symbol $|\Gamma_p|$. This is the size of the set $\Gamma_p = \{p | p \in_{def} \Gamma\}$. We will use the symbol O_{SLD} for complexity of computation of an SLD refutation from Γ in next paragraphs.

Now we can estimate the complexity of the probability function G as $O(O_{SLD} \cdot |r|)$. As it is obvious, if we restrict the form of a DMLP background theory Γ, the most important element in the given expression will be $|r|$. Therefore we can say that the final complexity of the probability function G is

$$O_G = O(|\Gamma_p| \cdot |r|)$$

When we speak about the estimative DMLP induction operator Ind_Ξ, which uses the evaluation function Ξ as the core of its calculation, we are in quite different position than in

[1]By this we mean programs without recursion. For more information on hierarchic logic programs refer to [**Llo 87**] page 110.

the case of the precise operator. This function does not need to walk through the database to test every tuple in the database r whether it belongs to the part of the database defined by the given clause or not. It only needs to compute a DMSLD refutation from $\Gamma_{base} \cup \Phi$, where Γ_{base} is the DMLP background base of an induced DMLP program Φ. Unlike the computation by the function G, the evaluation function Ξ needs to compute an estimation of the probability of the given expression from DMLP programs which are in this case Γ_{base} and Φ. The complexity of such computation is higher than the complexity of the computation of an SLD refutation from Γ. For computing any DMSLD refutation we need to enumerate all SLD refutations of the given expression and find the one, which maximizes its value of the probability. The most important element of this is the complexity of the computation of an SLD refutation. Again we will not even estimate complexity of it. We believe that under such strong syntactic constraints on a DMLP background theory Γ, as the constraint of hierarchy is, it is possible to treat the complexity of the function Ξ as

$$O_{\Xi} = O(c \cdot |\Gamma_p|) = O(|\Gamma_p|)$$

Now let us take a closer look on the generic algorithm FFC. As it is obvious, in the worst case it has to walk through all members of the set Γ^* of all expressions, in the form of conjunction, which are possible to produce from expressions of the form $p(x)$, where $p \in_{def} \Gamma$. Size of the set Γ^* is $|\Gamma^*| = n!$. Therefore an estimation of the complexity is in the worst case $O(n!)$. This is quite high value of complexity and shifts the algorithm FFC into the class of algorithms with very high complexity. Even we can say that the FFC problem is NP hard. In the worst case, there is no other way, than to enumerate all possible candidates to frequent conjunctions, however we feel that in an average case it is not that bad. We will not precisely estimate the complexity of the FFC algorithm, because it is not crucial for this work. We believe that by usage of heuristics for particular application it is possible to reduce the complexity of FFC. Even in some situations we can use user driven (or semi-automatic) proposing of candidates for new DMLP rules, or frequent conjunctions and by that reduce the complexity of FFC algorithm. We will discuss this topic later.

While the algorithm FFC uses the estimation function $Estimate_{conj}$ we need to involve also this function into our game. For each candidate for frequent conjunction we need to compute the value of $Estimate_{conj}$, therefore we can fix the complexity of FFC algorithms of both DMLP induction operators on

Ind_G	Ind_{Ξ}
$O^G_{FFC} = O(n! \cdot O_G) = O(n! \cdot \lvert\Gamma_p\rvert \cdot \lvert r\rvert)$	$O^{\Xi}_{FFC} = O(n! \cdot O_{\Xi}) = O(n! \cdot \lvert\Gamma_p\rvert)$

The algorithm CIR, is simpler than the FFC algorithm. It only computes for each frequent conjunction its probability. We meet here with the estimation function $Estimate_{rule}$, what is again the function based on either the probability function G in the case of precise DMLP induction operators, or the evaluation function Ξ in the case of an estimative approach to the DMLP induction. In fact the complexity of the CIR algorithm is depending on the number of frequent conjunctions and their sizes. The maximal possible size of the frequent conjunction is the same as the number of predicates defined in the given DMLP background theory Γ. It is equal to $|\Gamma_p|$. Number of frequent conjunctions is at most $|\Gamma^*| = n!$. However, in an average case this number will be much, much lower. Finally the algorithm must use the estimation function $Estimate_{rule}$ which depends on the complexity either of the function G, or Ξ. Therefore final complexity of the CIR algorithm for both DMLP induction operators will be:

Ind_G	Ind_Ξ
$O^G_{CIR} = O(n! \cdot \lvert\Gamma_p\rvert^2 \cdot \lvert r\rvert)$	$O^\Xi_{CIR} = O(n! \cdot \lvert\Gamma_p\rvert^2)$

In both operators we can use these specialised generic algorithms one after another or we can integrate the CIR algorithm into the FFC algorithm. Although it will not change the final complexity, by this we can speed up the whole process of the DMLP induction.

4.3. Hybrid approaches to DMLP induction

Likewise Mannila claims, naive implementations of generic algorithms usually leads to slow operations. Therefore, as it was already said, it is necessary to search for an appropriate heuristics which will improve the efficiency of the DMLP induction. Here we will give few naive proposals to improve it. We will not do more on this field, because it is not the main aim of this work.

While the usage of the evaluation function for the estimation $Estimate_{rule}$ can produce quite inaccurate, or too low, probabilities of DMLP clauses in the DMLP program Φ, we can try to mix both approaches to the DMLP induction to produce another DMLP induction operator. We will call it the hybrid DMLP induction operator and denote it by the symbol $Ind_{\Xi/G}$.

DEFINITION 4.3.1. Let r be a relational database, Δ the set of DMLP examples associated with r, Φ be a DMLP program with a DMLP background theory Γ. Let also G be the probability function and Ξ be the evaluation function Let finally A be a conjunction of predicates defined in Γ and $C = C_{head} \leftarrow C_{body}$ be a range restricted definite clause constructed from predicates defined in Γ.

We say that a DMLP induction operator $Ind_{\Xi/G}(r, \Gamma, G, \epsilon)$, which uses the generic skeleton, is *estimative* when it implements the estimation functions as follows

$$Estimate_{conj}(r, A) = \Xi(\Delta, \Gamma, \Phi, A)$$

$$Estimate_{rule}(r, C) = G(\Delta, \Gamma, \Phi, C)$$

By mixing the probability function G and the evaluation function Ξ into the one DMLP induction operator we have now an operator which produces DMLP programs with probabilities of clauses as precise as possible, still for proposing new clauses runs on the ground of the evaluation function Ξ. The complexity of this weak DMLP induction operator lies somewhere between complexities of operators Ind_G and Ind_Ξ. We can say that

$$O_{Ind_\Xi} \leq O_{Ind_{\Xi/G}} \leq O_{Ind_G}$$

Another approach to improve the efficiency and accuracy of DMLP induction operator is to replace the FFC algorithm by an intervention of the user. If the user will propose new clauses, or just frequent sets, the CIR algorithm will serve as an evaluator of such proposals. By this we can produce, in some applications, the DMLP induction operator which will be more efficient and more reliable in the process of DMLP induction.

Proof of the statement, that operator $Ind_{\Xi/G}$ is the correct weak DMLP induction operator would be straightforward and would be based on the proof of the theorem 4.1.8.

4.4. ϵ-tightness and DMLP induction

At the end of this chapter let us discuss the types of DMLP induction operators defined in subsections 4.1.3 and 4.1.4. Please note that theorems 4.1.8 and 4.1.11 are formulated only for weak DMLP induction operators. An ideal goal is to find an operator of DMLP induction, which will use the evaluation function Ξ and the theorem about ϵ-tightness for such an operator will be proved. Unfortunately we were not able to prove this theorem for the precise induction operator Ind_G nor for the estimative operator Ind_Ξ. Even we doubt whether it is possible. We suspect that some bad-looking result could be proven about impossibility of proving such theorem. This is the reason why we defined strong version of DMLP induction, but all our proposals for induction operators were based only on the weak version of the DMLP induction operator.

CHAPTER 5

Usage of DMLP

We developed the theory of Data Mining Logic Programs with the primary aim to use these programs as condensed representations. In the first sections of this chapter we will discuss exactly this topic. In the next sections we will more speculate. These sections will be quite informal and please treat them as such. However we feel that these speculations are not too crazy and maybe topics discussed there could be inspiring for some kinds of applications.

5.1. Condensed representations

As it was discussed in the subsection 2.2.5, one of the the biggest open issues in the field of data mining is the topic of condensed representations. As it was already said, condensed representation is a data object (structure) which represents the given underlying database itself, describes it approximately good and the most important point is, that it makes it possible to answer some probabilistic queries more efficiently than by looking into the database.

Here we give Heikki Mannila's definition of the condensed representation cited from [**Man 97b**]:

DEFINITION 5.1.1. Given a data collection $d \in \mathcal{D}$, and a class of patterns $\mathcal{P}$, a condensed representation for d and $\mathcal{P}$ is a data structure that makes it possible to answer queries in the form of *"How many times does $p \in \mathcal{P}$ occur in d?"* approximately correctly and more efficiently than by looking at d itself.

We think that the Data Mining Logic Programs are suitable candidate for condensed representations, because they describe the database according to some given DMLP background theory Γ, even it is possible to compute answers from DMLP programs to appropriate questions efficiently by using the evaluation function Ξ.

In fact, a set of DMLP clauses can be viewed as a set of sentences of the type: *"If the head of the clause holds, then the probability that the body of it holds, too, is at most the value of its label."* This allows us to chain such clauses into sequences (these are used in DMSLD refutations) of resolvents, which say something like: *"If the first member of the chain holds, then the probability that also the last member of it holds, too, is not greater than the product of labels in the chain."* As it is obvious, this probability for a single clause is similar to the confidence of an association rule used in the data mining. Confidence can be seen as the probability that the head of the clause holds when its body holds for sure.

The difference between these two terms is simple. In fact, both of them are conditional probabilities, but they are complementary to each other. Let us have a clause $C = A \leftarrow B$.

The confidence is represented by the conditional probability $P(A|B)$ and the probability of the given DMLP clause is $P(B|A)$. For the verification of this claim compare the equation $P(B|A) = \frac{P(A\cap B)}{P(A)}$ and the definition of probability of the DMLP clause $\frac{|A\wedge B(r,\overline{x_A}\cup\overline{x_B})|}{|r|}$. Even from Bayes theorem $P(B|A) = \frac{P(B)\cdot P(A|B)}{P(A)}$ we are able to express the confidence from the probability of DMLP clause. Note that values of probabilities $P(B)$ and $P(A)$ are known in the process of computation of the label of the DMLP clause by the evaluation function Ξ.

Many times in the previous chapter (3) we mentioned that the induction of DMLP programs with the evaluation function Ξ is probably more efficient than the induction of DMLP programs by usage of the probability function G. The main result is in the section 4.2 about the complexity of the DMLP induction operators. For clever implementations of generic algorithms, the most time-consuming part of them would be the raw walk through the database when we are using the probability function G in the core of the algorithm. With usage of the evaluation function Ξ, this element of the complexity can be reduced to the minimum at the cost of loss of precision. Still, such DMLP program is able to help in computation of the label of newly induced clause. This mechanism is described in the definition 3.3.23 and proven that it works in the theorem 3.3.26. This is exactly the task which has to be handled by condensed representations.

Note that such structures like condensed representations could be used to compress the information in the database. Similar topic for Stochastic Logic Programs is discussed by Muggleton in [**Mug 00**].

For these reasons we afford us to propose DMLP programs as a suitable candidate for condensed representations in data mining.

5.2. Probabilistic symbolic induction

As it was said in the introduction to this chapter, this section will be dedicated to an informal discussion of other possible applications of DMLP programs. All of these proposals will have one thing common. While DMLP programs can be seen as generalisation of logic programming, we can say that they are situated in the field of symbolic approaches to artificial intelligence. Such approaches have advantage of possibility of translation of their structures into the mathematical logic and then to the natural human language. Afterwards we are able to articulate them. Subsymbolic approaches usually do not have this property. This feature will be used in next discussions.

5.2.1. Extraction of rules from neural networks. Neural networks are biologically inspired data structures and algorithms which allow us to simulate kind of neural activity and by this help us to solve some non-trivial problems. We will not discuss in this work, details and the theory can be found in many books (e.g. [**Hri 98**]).

The main feature of neural networks is that we are able to learn them from examples. If we have for each input of the training set also the output of it, we can force the neural net to learn to compute for any similar input the same output. We can see neural networks as a data structures which contain *generalisation* of the training set. Learning of neural networks can be seen as the process of induction of this generalisation.

Neural nets give us quite good results in many fields of their application. Still we are not able to articulate, or explain why the net computed the result which it gave for given input. We just know that it is approximately corect. Our idea is to use the induction of DMLP programs for the same task, or in the task of extraction of rules from neural network.

Let us have a simple neural network, which works as a clasifier of input data into classes of outputs. For simplicity let input data be tuples over domains $D_1, \cdots, D_n$ and output be a binary vector with the only 1 in it. Let us have a relational database r which contains a training set as joined tuple of input columns and output part. If we have suitable DMLP background theory Γ, which allows a DMLP induction operator and an induced DMLP program to work with columns, then we are able to run an DMLP induction operator on r and Γ to produce a DMLP program Φ. This program will contain probabilistic statements in the form of clauses from which we would be able to extract partial information about probabilistic rules which hold in the network.

We can also set this idea on the top of its head by applicating it to some unknown neural net. As it was already said, neural network can be seen as a black box which *somehow* computes the result from the given input. Let us have some unknown neural network and the task is to find the result computed by it as an output. Again we can produce the set containing stochastic input data, let neural network compute the result for each member of it and finally let us run DMLP induction over this set of examples. The result of such induction will be some probabilistic description of relations between inputs and outputs. This may help by further investigation of the given unknown neural network.

Because of its properties, DMLP induction seem to be suitable to solve many tasks which are solved by neural networks, still while we did not develop any serious implementation of DMLP induction, we do not want to directly claim this. We only hope, that this approach can help in these tasks. Please note that this is only informal speculation.

5.2.2. Hidden Markov Models. As Muggleton in [**Mug 00**] claims, Stochastic Logic Programs were introduced originally as a way of lifting stochastic grammars to the level of first order Logic Programs. Later it was shown that SLPs can be used to represent also Hidden Markov Models and undirected Bayes' nets. While DMLP programs share many properties with SLPs, which were the inspiration for it, we afford us to claim that DMLP programs can be used in fields where Hidden Markov Models are used, too. Again, similary to neural networks, Hidden Markov Models are learned or constructed automatically from a training set to produce intended results.

Hidden Markov Models were successfully used for example in the field of speech recognition. Therefore we hope that DMLPs would have application in relative fields too.

Still note that this is only speculation, because, as we already mentioned above, there is no implementation of any DMLP induction operator, even we doubt whether such application will give something new in this field, or whether it will fasten some process. By the given speculation we just wanted to show, another possible field of application of DMLP.

CHAPTER 6

Conclusions

In the core of this work we proposed a framework called DMLP. It was strongly inspired by Stochastic Logic Programs, but the goal was to develop a structure suitable to serve as the condensed representation in inductive databases in the field of Data Mining together with the iterative mechanism of using these structures to induce more sophisticated condensed representations. By this we can generalise that the main theme of this work is usage of Logic Programming and approximative, or probabilistic reasoning in Data Mining.

This work is only student diploma thesis and as such it probably contains many bugs and has many limitations. Later we will discuss few open problems and directions of possible further development. The main doubt is usefulness of this work. Our feeling is that although this is an peripheral theme of the field of data mining, there are streams in the research around KDD and DM which have similar background and attack similar problems (especially works of Luc de Raedt, Heikki Mannila, Manfred Jaeger et al.).

6.1. Further development and open problems

Finally we give list of open topics raising from this work and possible further developments of it.

(1) Extend the DMLP theory to handle function symbols and more complicated domains (not only binary tables).
(2) The theory of DMLP works only with single relational table (database). One of simple extensions would be handling number of tables (databases).
(3) Revisions of DMLP programs can be interesting issue. If we would like to join two inductive databases (raw data + DMLP representation of the database) we need to join DMLP programs too.
(4) Develop theory and proove theorems about comprimation of the database by inducing DMLP program. This probably will be lossy compresion and there is a need to express the loss ratio (starting point should be Muggleton's work on U-learning in [**Mug -**]).

6.2. Contributions

At the end, let us finally summarize main contributions of this work. As the main result of this work we see the original framework for condensed representations in inductive databases which is able to hold sentences in the specific clone of logic programming language suitable to express approximative and statistical properties of the given relational database.

Glossary of used abbreviations and symbols

CIR	-	Computations of Induced Rules
DM	-	Data Mining
DMSLD	-	Data Mining SLD
DMLP	-	Data Mining Logic Program
FFC	-	Finding Frequent Conjunctions
FFP	-	Finding Frequent Patterns
LP	-	Logic Programming
KDD	-	Knowledge Discovery in Databases
SLD	-	Selection function Linear resolution for Definite clauses
SLP	-	Stochastic Logic Program
Δ	-	the set of DMLP examples
Γ	-	DMLP background theory
Γ_{base}	-	DMLP background base
Φ	-	induced DMLP program
r	-	database (relational table)
R	-	the relational schema
Ξ	-	evaluation function
G	-	probability function
Ind_{Ξ}	-	estimative DMLP induction operator
Ind_{G}	-	precise DMLP induction operator
$Ind_{\Xi/G}$	-	hybrid DMLP induction operator

Bibliography

[Agr 02] Agrawal, R. *Potentials and Challenges of Data Mining.* Invited talk at EDBT 2002, workshop DTDM-02, 2002

[Bac 90] Bacchus, F. *Probabilistic Belief Logic.* In proc. of ECAI-90, pages 59-64, 1990

[Hri 98] Hristev, R. M. *The ANN Book.* Edition 1, 1998

[Imi 95] Imielinski, T. Invited talk at KDD-95, 1995

[JMW 96] Jaeger, M., Mannila, H., Weydert, E. *Data Mining as Selective Theory Extraction in Probabilistic Logic.* In SIGMOD'96 Data Mining workshop, 1996

[Llo 87] Lloyd, J. W. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1987

[Man 97a] Mannila, H. *Inductive Databases and Condensed Representations.* In proc. International Logic Programming Symposium '97, pages 21-30, 1997

[Man 97b] Mannila, H. *Methods and Problems in Data Mining.* In proc. ICDT'97, 1997

[Mug 96] Muggleton, S. *Stochastic Logic Programs.* In proc. 5th International Workshop on Inductive Logic Programming, pub. DCS Katholieke Universiteit Leuven, ed. de Raedt, L., page 29, 1995

[Mug 00] Muggleton, S. *Learning Stochastic Logic Programs.* In proc. AAAI2000 Workshop on Learning Statistical Models from Relational Data, 2000

[Mug -] Muggleton, S. *Statistical Aspects of Logic Based Machine Learning.*

[MuR 94] Muggleton, S., de Raedt, L. *Inductive Logic Programming: Theory and Methods.* Journal of Logic Programming, vol. 19/20, pages 629-679, 1994

[Sef 00] Šefránek, J. *Inteligencia ako výpočet.* pub. IRIS, 2000