Discovery of Multivalued Dependencies from Relations

Iztok Savnik
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Am Flughafen 17, 79110 Freiburg, Germany
savnik@informatik.uni-freiburg.de

Peter A. Flach
Department of Computer Science, University of Bristol
Bristol BS8 1UB, United Kingdom
Peter.Flach@bristol.ac.uk

March 7, 2000

Abstract

Discovery of multivalued dependencies from database relations is viewed as a search in a hypothesis space defined according to the generalisation relationship among multivalued dependencies. Two algorithms for the discovery of multivalued dependencies from relations are presented. The top-down algorithm enumerates the hypotheses from the most general to more specific hypotheses which are checked on the input relation. The bottom-up algorithm first computes the invalid multivalued dependencies. Starting with the most general dependencies, the algorithm iteratively refines the set of dependencies to conform with each particular invalid dependency. The implementation of the algorithms is analysed and some empirical results are presented.

Keywords: Attribute dependency, multivalued dependency, database reverse engineering, and data mining.

1 Introduction

Dependencies between attributes of a database relation express the presence of structure in that relation, that can be utilised in the database design process. In particular, the existence of a multivalued dependency $X \rightarrow\!\!\!\rightarrow Y$ in a given relation $r(R)$, where $X, Y \subseteq R$ and $X \cap Y = \emptyset$, denotes that for each possible value of attributes X, there exist no associations between the values of attributes from Y and from $Z = R - X - Y$. As a consequence, the relation $r(R)$ can be decomposed into relations $r_1(XY) = \Pi_{XY}(r)$[1] and $r_2(XZ) = \Pi_{XZ}(r)$ without loss of information. The decomposition of r into r_1 and r_2 makes the internal structure of relation r explicit. Furthermore, the new representation requires less storage space than the complete relation r.

Traditionally, database dependencies were considered to be part of the data model provided by the database designer. However, they may also be retrieved from the extensional data. One reason for doing so can be that the data model, or parts of it, has been lost or is no longer accurate, so that some form of *reverse engineering* is required. Another reason may be that certain dependencies were not foreseen by the database designer, but do occur in practice. Once they have been discovered, they may be utilised for restructuring the database, as indicated above, but also for query optimisation. In this paper we address this problem of multivalued dependency discovery, understood as characterising the set of multivalued dependencies that are satisfied by a given collection of data.[2]

The paper presents two algorithms for discovery of multivalued dependencies from relations. The presented algorithms are based on the theoretical framework defined for knowledge discovery in relational databases by Mannila and Toivonen [9]. The top-down algorithm enumerates the hypotheses from the most general to more specific hypotheses. The hypotheses are verified on the complete relation. The bottom-up algorithm computes the set of valid multivalued dependencies using a previously elicited set of invalid multivalued dependencies. Starting with the set of most general dependencies, the bottom-up algorithm iteratively specialises this set in order to avoid contradiction with each particular invalid dependency.

The algorithms for the discovery of multivalued dependencies are based on our previous work on the discovery of database dependencies from relations presented

[1] $\Pi_S(r)$ is a standard projection operation which selects from r the values of attributes from S.

[2] We avoid using the ambiguous term 'dependency inference', which has been used in the literature both for dependency discovery and for the problem of constructing dependencies that are implied by given dependencies, which is not the problem we are dealing with in this paper.

in [5]. The present paper details the application of top-down and bottom-up algorithms for the discovery of multivalued dependencies. The contributions of this paper are: the definition of the hypothesis space for the discovery of multivalued dependencies; the introduction of the data structure which allows for efficient manipulation of the sets of multivalued dependencies; an improved procedure for the enumeration of hypotheses; and the empirical analysis of the developed discovery algorithms. A more detailed description of the relations between the work on the discovery of database dependencies presented in [5] and the work on the discovery of multivalued dependencies presented in this paper is given in Section 5.

1.1 Overview of the paper

Section 2 introduces multivalued dependencies, as well as a language providing an efficient representation. The generalisation/specialisation relationship among the sentences of the language is defined, and the concept of border of a theory is introduced.

In Section 3 we present algorithms for the enumeration of sentences, for testing the generalisation/specialisation relationship among sentences, and the top-down and bottom-up algorithms for the discovery of multivalued dependencies. Next, Section 4 presents some aspects of the implementation of the algorithms: the data structure for storing sets of sentences, the procedure for the enumeration of sentences, and some empirical results obtained by the prototype implementation of the algorithms.

Section 5 presents other work closely related to the discovery of multivalued dependencies, and compares the applications of the presented algorithms for multivalued dependencies and functional dependencies. Concluding remarks and some directions for further work are given in Section 6.

2 Preliminaries

This section presents the basic properties of multivalued dependencies, an efficient representation of multivalued dependencies, and the relationships among multivalued dependencies which are utilised in the induction algorithms. The following sub-section gives a formal definition of multivalued dependencies and introduces the concept of a dependency basis. Section 2.2 presents the multivalued dependencies in the framework of Model theory [2]. The concept of the border of a theory is defined and its role in the discovery of multivalued dependencies is presented.

2.1 Multivalued dependencies

Our notational conventions are close to [6]. A *relation scheme* R is an indexed set of *attributes* $A_1, \ldots, A_n$. Each attribute A_i has a *domain* D_i, $1 \leq i \leq n$, consisting of *values*. Domains are assumed to be finite or countably infinite. A *tuple* over R is a mapping $t : R \to \bigcup_i D_i$ with $t[A_i] \in D_i$, $1 \leq i \leq n$. A *relation on* R is a finite set of tuples over R, denoted $r(R)$. We will only consider finite relations. Any expression that is allowed for attributes is extended, by a slight abuse of symbols, to sets of attributes, e.g., if X is a subset of R, $t[X]$ denotes the tuple composed of values from the set $\{t[A] \mid A \in X\}$.

Let R be a relation scheme, let X and Y be subsets of attributes from R, and let Z denote $R - XY$. The expression $X \twoheadrightarrow Y$ is a *multivalued dependency* over R (*mvd* for short). Given an mvd $X \twoheadrightarrow Y$, the tuple t_3 *determined* by a pair of tuples t_1 and t_2 over R such that $t_1[X] = t_2[X]$ is defined by $t_3[XY] = t_1[XY]$ and $t_3[XZ] = t_2[XZ]$. Given a relation $r(R)$, a pair of tuples $t_1 \in r$ and $t_2 \in r$ *violates* a multivalued dependency $X \twoheadrightarrow Y$ if $t_1[X] = t_2[X]$ and the tuple determined by t_1 and t_2 is not in r. A relation $r(R)$ *violates* a multivalued dependency $X \twoheadrightarrow Y$ if some pair of tuples $t_1 \in r$ and $t_2 \in r$ violates $X \twoheadrightarrow Y$; t_1 and t_2 are called *positive witnesses*, and the tuple determined by them is called a *negative witness*. r *satisfies* an mvd if r does not violate it.

Thus, a relation r satisfies an mvd $X \twoheadrightarrow Y$ if for every pair of tuples $t_1 \in r$ and $t_2 \in r$ such that $t_1[X] = t_2[X]$ the tuple t_3 determined by t_1 and t_2 is in r as well. Note that by exchanging t_1 and t_2, there should also be a tuple $t_4 \in r$ with $t_4[XY] = t_2[XY]$ and $t_4[Z] = t_1[Z]$. Furthermore, notice that a pair of tuples can only violate an mvd in the context of a particular relation, because we need information about tuples not being in the relation.

Example 2.1 *Consider a relation describing weekly lectures. This relation satisfies a multivalued dependency from day of week to date: given the day of week, we can determine the set of dates on which the event occurs. For instance, if the Software Engineering course and the Artificial Intelligence course are both taught on a Wednesday during the fall semester, and there is an SE lecture on Wednesday October 6, while there is an AI lecture on Wednesday October 13, then there is also an SE lecture on the latter date and an AI lecture on October 6.*

Course	*Weekday*	*Date*
AI	*Wednesday*	*Oct 6*
SE	*Wednesday*	*Oct 6*
CS	*Thursday*	*Oct 7*
AI	*Wednesday*	*Oct 13*
SE	*Wednesday*	*Oct 13*
CS	*Thursday*	*Oct 14*

With Wednesdays we associate the values AI and SE for the Course attribute, and the values Oct 6 and Oct 13 for the Date attribute. With Thursdays we associate the value CS for the Course attribute, and the values Oct 7 and Oct 14 for the Date attribute. If we fix the Weekday attribute, there are no associations between the Course attribute on one hand and the Date attribute on the other. □

Given a set of multivalued dependencies $\mathcal{M}$ which are valid in a relation $r(R)$, additional dependencies valid in $r(R)$ can be derived from $\mathcal{M}$ using the *inference axioms* for multivalued dependencies. The axioms are as follows [6]: M1) reflexivity, M2) augmentation, M3) additivity, M4) projectivity, M5) transitivity, M6) pseudo-transitivity, and M7) complementation axioms. Only the first four axioms which are needed in the sequel are presented here; the complete definition of the axioms as well as a completeness theorem for the inference axioms can be found in [6, 1, 14].

M1) Reflexivity: $X \rightarrow\!\!\rightarrow X$
M2) Augmentation: $\{X \rightarrow\!\!\rightarrow Y\} \wedge Z \subseteq R \models XZ \rightarrow\!\!\rightarrow Y$
M3) Additivity: $\{X \rightarrow\!\!\rightarrow Y, X \rightarrow\!\!\rightarrow Z\} \models X \rightarrow\!\!\rightarrow YZ$
M4) Projectivity: $\{X \rightarrow\!\!\rightarrow Y, X \rightarrow\!\!\rightarrow Z\} \models X \rightarrow\!\!\rightarrow Y \cap Z$

The augmentation, additivity and projectivity axioms provide the basis for an efficient representation of the set of multivalued dependencies. Let F be a set of multivalued dependencies over the schema R. For a given set of attributes X, we can construct a set $G = \{Y \mid F \models X \rightarrow\!\!\rightarrow Y\}$. The set G includes all valid right-hand sides Y of the multivalued dependencies $X \rightarrow\!\!\rightarrow Y$ which can be derived from F using the axioms just presented.

According to the projectivity axiom, each pair of sets $Y_1, Y_2 \in G$ can be replaced by the sets $Y_1 \cap Y_2$, $Y_1 - Y_2$, and $Y_2 - Y_1$. Iterative application of the above rule on the set G yields the *minimal disjoint set basis* of G denoted $mdsb(G)$ [6]. By using the additivity axiom, each $Y_i \in G$ can be now expressed as a union of some elements of $mdsb(G)$. The *dependency basis* of X with respect to a set of multivalued dependencies F is $DEP(X) = \{Y_1, \ldots, Y_m\}$ where $Y_j \in mdsb(G)$.

Example 2.2 *In Example 2.1 we have $R = \{Course, Weekday, Date\}$ and $Weekday \rightarrow\!\!\!\rightarrow Date$, hence $DEP(Weekday) = \{Course, Date\}$. Notice that $X \rightarrow\!\!\!\rightarrow A$ is trivially valid for all $A \in X$, but elements of X are normally not included in $DEP(X)$.*

The dependency basis is used for representing multivalued dependencies in the algorithms for discovery of multivalued dependencies from relations. The multivalued dependencies defined over the relational schema R are represented by sentences of the form $(X, DEP(X))$ where $X \subset R$ and $DEP(X)$ is the dependency basis of X.

The reasons for using dependency bases as representation of multivalued dependencies are the following. Given a set of attributes X and the set of multivalued dependencies F which are valid in r there exists a single dependency basis which represents all dependencies $X \rightarrow\!\!\!\rightarrow Y$ which are entailed by F. Furthermore, the representation of F using dependency bases is efficient for verifying whether a multivalued dependency $X \rightarrow\!\!\!\rightarrow Y$ is entailed by F. The test can be performed by checking if the right-hand side of the dependency Y can be expressed as the union of some sets from $DEP(X)$ [1]. Finally, the dependency bases turned out to be a suitable representation of multivalued dependencies for the definition of the enumeration procedure in discovery algorithms.

2.2 Theory of a relation

Let $r(R)$ be a relation, and let $S = (X, DEP(X))$ be a sentence as defined above. The *selection predicate* $q(r(R), S)$ is a truth function which, given a relation r and a sentence S, returns $True$ if S holds in r and $False$ otherwise. The selection predicate is based on the definition of the multivalued dependencies presented in Section 2.1. A sentence $S = (X, DEP(X))$ is true in r, or $q(r, S) = True$, if every dependency $X \rightarrow\!\!\!\rightarrow Y$ where $Y \in DEP(X)$ is true in relation r. The truth of a sentence S in r implies the truth of all multivalued dependencies which can be derived from the set $\{X \rightarrow\!\!\!\rightarrow Y_i \mid Y_i \in DEP(X)\}$ using the inference axioms for multivalued dependencies.

The set of all sentences which are true in relation $r(R)$ is called the *theory* of relation $r(R)$ with respect to the language L of multivalued dependencies and the selection predicate q [9]. Formally, the theory of r with respect to L and the selection predicate q is $\mathcal{T}(L, r, q) = \{S \mid S \in L \wedge q(r, S) = True\}$.

The sentences of the theory are ordered by the partial ordering relationship $\preceq$ among the sentences of L. This relationship $\preceq$ plays an important role in the design of the algorithms for the induction of multivalued dependencies: it is used for enumerating hypotheses (sentences), and it serves as a means for selecting the minimal set of sentences equivalent to the theory $\mathcal{T}(L, r, q)$ which is maximally consistent.

Before presenting the partial ordering relationship $\preceq$, we present the relationship *is_refinement_of* among the dependency bases. Suppose the dependency bases DB_1 and DB_2 are given. The dependency basis DB_1 *refines* the dependency basis DB_2 if each element of DB_2 is the exact union of some elements from DB_1. A sentence $S_1 = (X, DEP(X))$ is more general than a sentence $S_2 = (Y, DEP(Y))$, written $S_1 \preceq S_2$, if the following two conditions are met: (1) $X \subseteq Y$, and (2) $DEP(X) \setminus Y$ refines $DEP(Y)$, where $DEP(X) \setminus Y = \{Z_i \mid X_i \in DEP(X) \wedge Z_i = X_i - Y\}$. We now present some examples of sentences related by $\preceq$.

Example 2.3 *Let $r(R)$ be a relation over the schema $R = ABCDEFG$.*

$(AB, \{C, DE, FG\}) \preceq (AB, \{DE, CFG\})$
$(A, \{B, CD, EF, G\}) \preceq (AB, \{CD, EF, G\})$
$(A, \{B, C, D, EFG\}) \preceq (ADF, \{BC, EG\})$ $\square$

The partial ordering relationship $\preceq$ is defined in accordance with the semantic implication relationship. The first condition of the above definition resembles the augmentation axiom (M2). The second condition of the definition implements the additivity axiom (M3). If a sentence S_1 is more general than a sentence S_2, or $S_1 \preceq S_2$, then S_2 is entailed by S_1; moreover, every sentence which can be derived from S_2 can also be derived from S_1. Therefore, the generalisation relationship $\preceq$ is monotonic: $q(r, S_1) = True \wedge S_1 \preceq S_2 \Longrightarrow q(r, S_2) = True$.

The relationship $\preceq$ is used to define a more optimal representation of the theory of a relation r. The sentences from the theory, which can be derived from a more general sentence from the theory, can be eliminated without loss of information. If all sentences for which there exists a more general sentence in the theory are eliminated, then the remaining set includes only the most general sentences of the theory — this set is called the *positive border* of the theory [9]. Formally, the positive border of theory $\mathcal{T}(L, r, q)$ is $\mathcal{B}^+(\mathcal{T}) = \{S \mid S \in \mathcal{T} \wedge \neg\exists S_1(S_1 \in \mathcal{T} \wedge S_1 \preceq S)\}$.

Similarly, the sentences which are false in r can also be represented by a smaller set of false sentences. Before presenting the details, some properties of the false sentences are given. First, the generality relationship $\preceq$ has in the case of false sentences a different interpretation. If the sentence S_1 is more general than the sentence S_2 and the sentence S_2 is false in r, then we can conclude that S_1 is also false in r. In general, all sentences, which are more general than a statement which is false in r, are also false.

Let $\mathcal{T}_n$ denote the set of all sentences from L which are false in r with respect to q. If all sentences from $\mathcal{T}_n$ for which there exists a more specific sentence in $\mathcal{T}_n$ are removed, the remaining set of sentences includes only the most specific false statements. This set is called the *negative border* of the theory, formally: $\mathcal{B}^-(\mathcal{T}) = \{S \mid S \in \mathcal{T}_n \wedge \neg\exists S_1(S_1 \in \mathcal{T}_n \wedge S \preceq S_1)\}$.

The union of the positive and the negative borders is called the *border of a theory*, denoted $\mathcal{B}(\mathcal{T})$. In short, the border of the theory includes the most specific sentences which are false, and the most general true sentences. Note also that the border consists of those sentences whose specialisations are all true, and whose generalisations are all false.

The problem of the discovery of multivalued dependencies from relations can now, in the framework of the above definitions, be seen as the computation of the positive border of a theory. The following section describes two algorithms for this computation.

3 Algorithms

In this section we present the top-down and bottom-up approaches to the computation of multivalued dependencies valid in the input relation. The top-down algorithm for the computation of the positive border of the set of valid dependencies starts with the set of most general multivalued dependencies. The set is specialised until the most general valid dependencies from the positive border are reached. The bottom-up algorithm starts with the computation of the negative border of the set of invalid dependencies by eliciting false dependencies directly from the input relation. It computes the positive border by enumerating the minimal specialisations of the sentences which form the negative border.

Before presenting the details of the algorithms for induction of multivalued dependencies, we present the procedure for the enumeration of sentences, and the procedure for testing the relationship $\preceq$ between the sentences, which are both employed in the top-down and bottom-up algorithms.

3.1 Enumeration of sentences

The enumeration of sentences is based on the lattice induced by the relationship $\preceq$. The most general sentence in the lattice defined for the schema R is the sentence $(\emptyset, R)$. Specialisation of sentences is achieved by using the following two rules. Let $S = (X, \{X_1, \ldots, X_n\})$, where $X \subseteq R$ and $X_i \subseteq (R - X)$ are non-overlapping sets, be a sentence defined under the schema R. Specialisation of S, denoted S', can be one of the following.

1. Rule: $S' = (Y, \{X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_n\})$, where $Y = X \cup X_j$
2. Rule: $S' = (X, \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_{j-1}, X_{j+1}, \ldots, X_{n+1}\})$, where $X_{n+1} = X_i \cup X_j$

In the first case, the set of attributes X is joined with an element (a set of attributes) of $DEP(X)$ yielding a new set of attributes Y as the first component of S'. The second rule states that two arbitrary elements X_i and X_j of the dependency basis $DEP(X)$ can be joined, yielding a set of attributes X_{n+1}, which is a new element of $DEP(X)$ in the sentence S'. If the sentence $(X, \{X_1, \ldots, X_n\})$ is valid in a relation r, then the above two rules generate, in accordance with Axioms M2 and M3, other sentences valid in r. Each S' thus generated is called an *immediate specialisation* of S; conversely, S is called an *immediate generalisation* of S'.

3.2 Testing relationship $\preceq$

Let S_1 and S_2 be sentences defined under the schema R. The relationship $S_1 \preceq S_2$ can be tested using the function **MGE** (More General or Equal) presented in Algorithm 1.

Note that the set D used in lines 3-6 corresponds to $DEP(X) \backslash Z$ as defined in Section 2.2. The function `refines(`DB_1`,`DB_2`)` at line 7 is used to check if the relationship *is_refinement_of* holds between DB_1 and DB_2. As presented in Section 2.2, DB_1 *refines* DB_2 if each element of DB_2 is the exact union of some elements from DB_1. For instance, the dependency basis $\{\{A_1\}, \{A_2, A_3\}, \{A_4\}, \{A_5\}\}$ is a refinement of the dependency basis $\{\{A_1, A_2, A_3\}, \{A_4, A_5\}\}$.

Input: Sentences $S_1 = (X, \{X_1, \ldots, X_n\})$ and $S_2 = (Y, \{Y_1, \ldots, Y_m\})$.
Output: $True$ if $S_1 \preceq S_2$, $False$ otherwise.
Method:

```
begin
    if (X ⊆ Y ) then
        Z = Y − X; D = ∅;
        foreach ( X_i ∈ DEP(X) ) do
            add (X_i − Z) to D;
        od;
        if (refines( D, DEP(Y) )) then
            return True and exit;
        fi;
    fi;
    return False;
end;
```

Algorithm 1: Testing relationship $\preceq$

3.3 Top-down algorithm

The top-down algorithm uses the enumeration of sentences as presented at the beginning of this section. The validity of sentences is determined using the function $q(r, s)$ which checks if a sentence s is valid in the relation r with respect to the definition of multivalued dependencies given in Section 2.1.

The top-down algorithm is presented as Algorithm 2. The variable $\mathcal{B}^+$ denotes a set of sentences which is used to gather the sentences from the positive border as they are generated by the algorithm. The variable $\mathcal{H}$ includes the hypotheses. Initially, the set includes the most general sentence. Later $\mathcal{H}$ is incrementally refined as the algorithm considers more specific hypotheses (line 10). Finally, the immediate specialisations of a sentence S which are generated in line 9 are those specialisations which can be obtained by a single invocation of either Rule 1 or Rule 2 presented at the beginning of this section.

3.4 Eliciting the negative cover

The bottom-up algorithm differs from the top-down algorithm in that it first constructs a set of invalid sentences, as follows. Let $r(R)$ be a relation with the

Input: A relation $r(R)$ defined over a schema R.
Output: A set of most specific sentences valid in $r(R)$.
Method:

```
begin
    B+ = {};
    H = {(∅, R;
    while ( H not empty ) do
        pick S from H;
        if (q(r, S) = True ) then
            add S to B+;
        else
            H_S = { all immediate specialisations of S };
            H = H ∪ H_S;
        fi;
    od;
end;
```

Algorithm 2: Top-down algorithm

schema R. We investigate each pair of tuples in r, and construct from them the most specific invalid sentences which are contradicted by that pair of tuples. The most specific invalid sentences are of the form $(X, \{Y_1, Y_2\})$ where $Y_1, Y_2 \subset R - X$ and the dependencies $X \rightarrow\!\!\!\rightarrow Y_1$ and $X \rightarrow\!\!\!\rightarrow Y_2$ are not valid. The sentences which are more specific with regards to the ordering defined by the procedure for the enumeration of sentences in Section 3.1 are trivial, while all more general invalid sentences have to be the generalisations of some of the above most specific invalid sentences.

The set of invalid sentences which is constructed after having dealt with each pair of tuples in this way, includes the most specific sentences which are false in r. We call this set the *negative cover*. While logically equivalent to the negative border, it does not necessarily contain the same set of sentences: in particular, it may contain several sentences with the same first component. However, compressing the negative cover would be an expensive operation, and is not needed for the correctness of the bottom-up algorithm. For this reason, the negative cover is used instead of the negative border.

Let us explain the algorithm for eliciting negative cover presented as Algorithm 3 in more detail. The set X includes the attributes on which t_1 and t_2

Input: A relation $r(R)$ defined over a schema R.
Output: The negative cover $\mathcal{B}^-$ of a set of sentences which are false in $r(R)$.
Method:

1. **begin**
2. $\quad \mathcal{B}^-$ = $\{\}$;
3. $\quad$ **foreach** ($t_1, t_2 \in r$) **do**
4. $\qquad X = \{A \mid A \in R \wedge t_1[A] = t_2[A]\}$; $Y = R - X$;
5. $\qquad$ **foreach** ($Z \subseteq Y$) **do**
6. $\qquad\quad$ **if** ($\neg\exists t_3, t_4 (t_3, t_4 \in r \wedge ((t_1[X] = t_3[X] \wedge t_1[X] = t_4[X]) \Longrightarrow$
7. $\qquad\qquad (t_3[Y] = t_1[Y] \wedge t_3[Y\text{-}Z] = t_2[Y\text{-}Z] \wedge$
8 $\qquad\qquad t_4[Y] = t_2[Y] \wedge t_4[Y\text{-}Z] = t_1[Y\text{-}Z]$)) **then**
9. $\qquad\qquad S = (X, \{Z, Y - Z\})$;
10. $\qquad\qquad$ **if** (S is not covered by $\mathcal{B}^-$) **then**
11. $\qquad\qquad\quad$ add S to $\mathcal{B}^-$;
12. $\qquad\qquad\quad$ remove all generalisations of S from $\mathcal{B}^-$;
13. $\qquad\qquad$ **fi**;
14. $\qquad\quad$ **fi**;
15. $\qquad$ **od**;
16. $\quad$ **od**;
17. **end**;

Algorithm 3: Eliciting negative cover

agree, and the set Y includes the attributes on which t_1 and t_2 do not agree. Therefore, any dependency $X \twoheadrightarrow Z$ with $Z \subseteq Y$ is false, if r does not contain two additional tuples t_3 and t_4 with the properties expressed in lines 6-8. For any such dependency $X \twoheadrightarrow Z$, a sentence $(X, \{Z, Y - Z\})$ is added to the currently discovered negative cover $\mathcal{B}^-$ (line 11) if $\mathcal{B}^-$ does not cover S, that is, if $\mathcal{B}^-$ does not include dependencies which are more specific than S (line 10). Finally, the generalisations of S are removed from the negative cover.

3.5 Bottom-up algorithm

The bottom-up algorithm is an iterative algorithm: it starts with the set of most general sentences and specialises it in each step of the iteration to conform with one false dependency from the negative cover. Therefore, the main loop of the algorithm iterates through the sentences comprising the negative cover. In

Input: A relation $r(R)$ defined over a schema R.
Output: Positive border $\mathcal{B}^+$ of the theory of $r(R)$.
Method:

```
1.  begin
2.      compute B⁻ using Algorithm 3;
3.      B⁺ = {(∅, R)};
4.      foreach ( S_n ∈ B⁻ ) do
5.         foreach ( S_p ∈ B⁺ contradicted by S_n ) do
6.            delete S_p from B⁺;
7.            foreach ( S'_p which is the closest specialisation
8.                        of S_p not contradicted by S_n ) do
9.               if ( S'_p not covered by B⁺ ) then
10.                  add to B⁺;
11.              fi;
12.           od;
13.        od;
14.     od;
15.  end;
```

Algorithm 4: Bottom-up algorithm

each step, the positive border is refined by specialising the sentences which are contradicted by the currently considered invalid sentence from the negative cover.

Let us present some details of the bottom-up algorithm which is presented as Algorithm 4. Firstly, searching the sentences S_p in $\mathcal{B}^+$ contradicted by S_n (line 5) can be seen as searching the sentences in $\mathcal{B}^+$ which are more general than S_n. As will be pointed out later, this procedure has to be implemented efficiently to be able to use the bottom-up algorithm for larger relations. Next, the sentences S'_p in line 7 are the closest specialisations of S_p which are not contradicted by S_n, that is, each S'_p is more specific than S_n, and at least one of the immediate generalisations of S'_p (those sentences that can be obtained by a single invocation of the inverse rules of Rule 1 and Rule 2) is contradicted by S_n. Finally, S'_p is covered by $\mathcal{B}^+$ if a more general sentence than S'_p exists in $\mathcal{B}^+$.

4 Implementation

This section presents some implementation details of the top-down and bottom-up algorithms. The data structure for efficiently representing sets of statements is described in the following section. The methods used for the optimisation of the enumeration process are described in Section 4.2. Finally, some results of experiments with the prototype implementation of the algorithms are presented in Section 4.3.

4.1 Representing sets of sentences

The representation of sets of sentences is important for an efficient implementation of the previously presented algorithms for discovery of multivalued dependencies. Such a data structure must allow efficient implementation of the operations add, delete, membership test, and generalised membership operations. Here, the first three operations have the usual semantics. The generalised membership operations search for a sentence which is either more specific than a given sentence S, or more general than S. Furthermore, since the negative and positive borders can contain large number of sentences, the data structure used for the representation of the sets of dependencies should be memory-efficient.

A data structure called MVD-tree is defined for representing sets of sentences as defined in Section 2. MVD-tree is based on the data structure for the representation of the sets of functional dependencies [13]. It meets all of the above stated requirements: it provides an efficient implementation of the above mentioned operations, and it takes advantage of the repeating patterns which appear in sentences to reduce the space needed to store a set of sentences. Let us first present an antecedent tree which is later used in the definition of the MVD-tree. Suppose that the attributes from R are totally ordered.

Definition 4.1 (Antecedent tree) *Given a relation scheme R, an* antecedent tree *over R is a tree with the following properties:*

1. *every node of the tree, except the root node, is an attribute from R;*
2. *the children of the node A_i are higher attributes; and*
3. *the children of the root are all attributes.* □

MVD-tree is an antecedent tree where the nodes of a tree include two labels. The label G of a node A includes a sentence which is either more general or more specific than all sentences represented by a subtree rooted at A. This label is used for the efficient realization of the generalised membership operations. The second label, denoted E, is used for the representation of the dependency bases of sentences stored in the MVD-tree. MVD-tree is defined as follows.

Definition 4.2 (MVD-tree) *Given a relation scheme R and a set of sentences $\mathcal{S}$ over R, the MVD-tree of $\mathcal{S}$ is defined as follows:*

- *for every sentence $(X, DB) \in \mathcal{S}$ there exists a path from the root of MVD-tree to the node A; the path includes all attributes from X and A corresponds to the highest attribute from X,*
- *the label G of each tree node A stores a sentence which is either the least upper bound or the greatest lower bound of all sentences represented by a subtree rooted at A, and*
- *the label E of each node A stores a set of dependency bases DB_i of sentences (X, DB_i) such that the path corresponding to X ends in the node A.*

The above definition provides two possible values of the label G. Suppose an MVD-tree T and a sentence $S = (X, DB)$ are given. Firstly, the value of G-labels of nodes A that constitute T can be the greatest lower bound (glb) of sentences represented by the subtree rooted at A. In this case, T is prepared for efficiently searching for a more specific sentence than a given sentence S. Secondly, the value of G-labels of nodes A can be the least upper bound (lub) of sentences represented by the subtree rooted at A. In this case, MVD-tree T is prepared for searching for a more general sentence than S in T.

Let us now present the operation which adds a new sentence $S = (X, DB)$ to an MVD-tree and, through this, illustrate the roles of the labels G and E. As defined above, S is in an MVD-tree represented as a path comprised of the attributes from X. Each G-label of nodes on the path is updated when a new sentence is added to the tree. In particular, the value of label G defined for a node A is replaced by a sentence which is either lub or glb of sentence S and the previous value of G for the node A. In this way, each G-label of nodes A on the path corresponding to X includes a sentence which bounds all sentences represented by a subtree rooted at A. This property of the MVD-tree is effectively used by the generalised membership operations. The E-labels of all nodes from the path representing S, except the last node from the path, are not affected by the addition of a sentence S to the MVD-tree. The dependency basis DB of S is

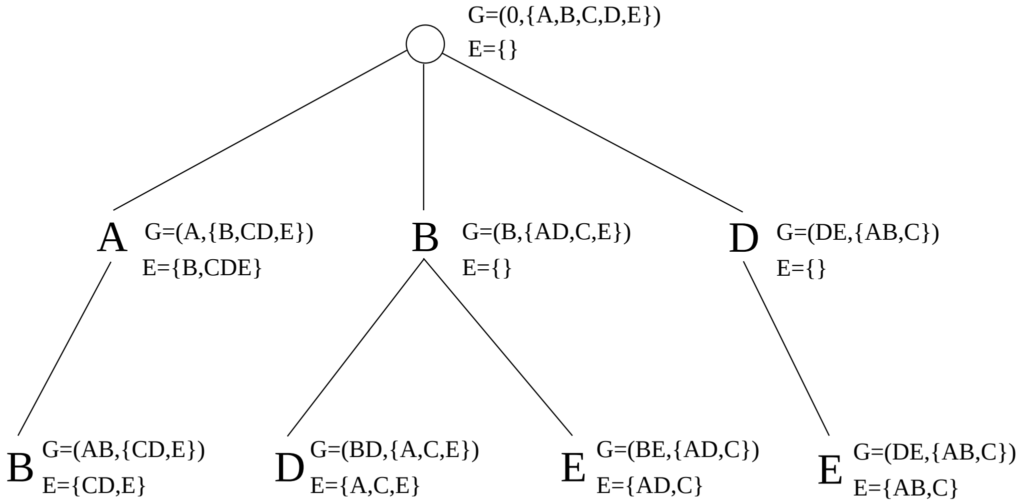

Figure 1: An example of MVD-tree.

added to the E-label of the last node of path that corresponds to X, indicating the last node of the sentence S, and storing the second component of S.

Example 4.1 *An example of an MVD-tree is presented in Figure 1. Suppose a relation r has the schema $R = ABCDE$. MVD-tree presented in Figure 1 stores the following sentences:* $(AB, \{CD, E\})$, $(A, \{B, CDE\})$, $(BD, \{A, C, E\})$, $(BE, \{AD, C\})$, *and* $(DE, \{AB, C\})$. $\Box$

As already mentioned above, the MVD-tree allows for the efficient implementation of the usual membership operation, and the generalised membership operations. Let us now sketch the realization of these operations. Suppose a sentence $S = (X, DB)$ and an MVD-tree T are given. The operation which verifies if S is an element of T is realized by a simple traversal starting at the root of T, following the attributes which constitute X, and finally checking if the dependency basis DB is included in the value of the label E of the node representing the highest attribute from X.

The generalised membership operations are used for searching a more specific or a more general sentence than a given sentence S in a MVD-tree T. For the former operation the labels G of nodes are computed using the operation glb. The operation which searches in T for a more specific sentence than S uses label G as a guide. Since the labels G of nodes A include the greatest lower bound of sentences represented by a subtree rooted at A, only the subtrees of T where S

is not more specific than the value of G need to be considered. In the case that S is more specific than G of a node A, then the subtree rooted at A does not include a more specific sentence than S. Furthermore, search is constrained by the contents of the set X. Namely, each path which is visited by the procedure and which ends by a node A, has to include all attributes from X which are smaller than or equal to A.

The operation for searching a more general sentence than $S = (X, DB)$ in an MVD-tree T is defined similarly to the above operation. For this operation the labels G of nodes A which constitute the tree T include the least upper bound of sentences represented by subtree rooted at A. The operation for searching a more general sentence than S in T considers the following sentences. Firstly, only those sentences need to be considered which are represented in subtrees of T where the sentence S is not more general than the label G. In the case that S is more general than G than the subtree rooted at A can not include more general sentences. Secondly, the sentences from T that need to be considered by the operation include only those sentences which are represented by a path Y such that $X \subseteq Y$.

4.2 Controlling the enumeration of sentences

The sentences over a relational schema R are ordered by the is-more-general relationship $\preceq$ which serves as a basis for the definition of the enumeration of sentences. Let us now consider the representation of sentences and the enumeration of sentences in more detail.

We assume some total ordering over the attributes of schema R. Furthermore, we assume the existence of a special attribute τ which is smaller than every other attribute. A sentence $S = (X, \{X_1, \ldots, X_n\})$ can now be written as a set of mutually disjunctive sets of attributes from R: $S' = \{X', X_1, \ldots, X_n\}$ where $X' = X \cup \{\tau\}$. In addition, the representation of sentences meets the following requirements. First, the attributes of each set from S' are written in ascending order, and second, the sets which constitute S' are written with respect to the ascending order of the first attribute from the set. For example, a sentence $(\{B\}, \{\{D, F\}, \{A, C\}, \{E\}\})$ over the schema $R = ABCDEF$ can be written as $\{\{\tau, B\}, \{A, C\}, \{D, F\}, \{E\}\}$. Finally, each sentence includes a *marker* which denotes the currently active set in a sentence. In the following examples marked sets are underlined. The purpose and the use of markers will be presented through the algorithm for the enumeration of sentences.

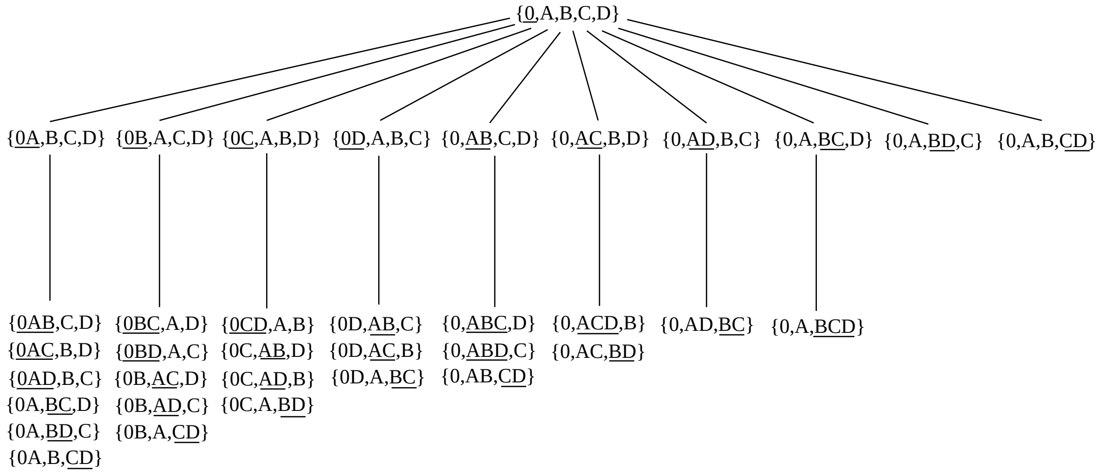

Figure 2: Sentences defined over the schema $R = ABCD$.

Example 4.2 *Suppose that the sentences are defined over the relation schema $R = ABCD$. The complete set of sentences over the schema R are presented in Figure 2. The sentences are arranged in three levels. The first level includes only the most general sentence which is composed of sets including single attributes. The first set of most general sentence includes only the special attribute τ which is denoted as 0. The second level includes the specialisations of the most general sentence $\{0, A, B, C, D\}$. The third level includes the specialisations of the sentences at the second level. Note that a column of sentences at the third level includes the specialisations of the sentence placed above the column (at the second level). The specialisations of the sentences at the third level are trivial sentences, which are not indicated. Finally, note that each sentence in the third level is a specialisation of two or three sentences from the second level. For instance, the sentence $\{0BD, A, C\}$ is a specialisation of the sentences $\{0B, A, C, D\}$, $\{0D, A, B, C\}$ and $\{0, A, BD, C\}$. The exact meaning of the lines between the sentences in Figure 2 are described below.* $\square$

The above example shows that a simple top-down enumeration of sentences over a schema R using the two rules presented in Section 3 would (in the case that R includes more than a few attributes) result in a large number of repeated sentences. Each of the repeated sentences would either have to be tested against the complete relation (Algorithm 3), or it would have to be tested against the negative cover of the theory (Algorithm 4). In any case, such enumeration significantly affects the performance of the previously presented algorithms. In order to avoid repeated enumeration, Algorithm 5 is used.

Input: A sentence $S = \{X_0, \ldots, \underline{X_i}, \ldots, X_n\}$ defined under relation $r(R)$.
Output: Sentences which are more specific than S.
Method:

```
1.  procedure enum ( sentence S );
2.  begin
3.      i = S.marker;
4.      foreach ( j ∈ [i..n] ) do
5.          foreach ( k ∈ [(j+1)..n] ) do
6.              if ( last(X_j) < first(X_k)) then
7.                  Y = X_j ∪ X_k;
8.                  S = {X_0, ..., X_{j-1}, _Y_, X_{j+1}, ...};
9.                  output( S );
10.                 enum( S );
11.             fi;
12.         od;
13.     od;
14. end;
```

Algorithm 5: Enumeration of sentences

To enumerate all sentences defined under the schema R the above procedure is called by taking the most general sentence as a parameter. The index of the marked element (active set) of a sentence S is denoted as S.`marker` (line 3). Next, notice the use of functions $first(X)$ and $last(X)$ in the algorithm: the function $first(X)$ returns the first element of the set X and the function $last(X)$ yields the last attribute from X. Algorithm 5 enumerates the sentences in a top-down manner. The specialisations of S are formed by joining two sets of attributes from S (line 7). The pairs of sets that are joined include all different pairs selected from the sets which are higher than the marked set and including the marked set (lines 4 and 5). Furthermore, pairs are joined only if the last attribute of the first set precedes the first attribute of the second set (line 6). The set which is formed by joining two sets becomes the active set.

Example 4.3 *Figure 2 presents the enumeration of the sentences defined under the schema $R = ABCD$. The most general sentence is $\{\tau, A, B, C, D\}$. The algorithm enumerates the statements in a top-down manner; the left branch of the tree is visited first. The lines between the sentences denote the flow of Algorithm 5 – for instance, after enumerating the sentence $\{0A, B, C, D\}$, the algorithm pro-*

Domain	$\|r\|$	$\|R\|$	n_{neg}	n_{pos}	t_{td}	$t_{bu/neg}$	$t_{bu/pos}$
Iris	150	5	12	4	2	510	1
Lenses	24	6	8	2	0.5	11	3
Led	100	8	10	5	56	668	546
Car	20	7	9	2	41	3	28
Car	50	7	14	2	174	55	39
Car	100	7	21	0	859	258	36
Bupa	20	7	62	15	1	47	6
Bupa	50	7	57	16	2	251	10
Bupa	100	7	75	20	3	1143	17
Abalone	20	9	223	11	23	840	843
Abalone	50	9	367	29	58	5646	637
Abalone	100	9	415	29	135	13875	1124

Table 1: Empirical results with the program `mdep`

ceeds with the enumeration of its specialisations $\{0AB, C, D\}$, $\{0AC, B, D\}$, *etc.*
□

4.3 Empirical analysis

The algorithms for discovery of multivalued dependencies were implemented using the Sicstus Prolog programming language in a program called `mdep`. The relations are represented in `mdep` as Prolog ground facts. The MVD-tree is represented by dynamic predicates. Each node n of the MVD-tree is stored as a predicate $tree(X, G, E)$ where the predicate name $tree$ denotes the name of a tree, the argument X represents the path from the root to the node n, G stores the least upper bound or the greatest lower bound of the sentences whose first component subsumes X, and E includes a list of dependency bases of sentences which end in n. The trees can be traversed by simply adding or removing the last attributes to/from X, progressing in this way downwards or upwards in the tree, respectively.

The experiments with `mdep` presented in Table 1 were done on a Sun station ULTRA 1. The datasets which were used in the experiments are available at UCI Machine learning repository [10]. In the case of the datasets `Car`, `Bupa` and `Abalone` we use randomly selected subsets of the original datasets. For each experiment we specify the name of the relation (dataset) $r(R)$, the number of tuples in relation $|r|$, the number of attributes of relation $|R|$, the number of sentences in the negative cover n_{neg}, the number of sentences in the positive border n_{pos}, the time (in seconds) required to compute the positive border using the top-down algorithm t_{td}, the time required to compute the negative cover t_{neg}, and the positive border t_{pos} using the bottom-up algorithm.

We will now comment on the results of the experiments presented in Table 1. The top-down algorithm outperforms in most cases the bottom-up algorithm. This is primarily due to the computation of the negative cover (Algorithm 3). This algorithm constructs the set of most specific invalid sentences for each particular pair of relation tuples. Let $r(R)$ be a relation where $|r(R)| = n$ and $|R| = k$. For each particular pair of tuples t_1 and t_2 which agree on the values of attributes from X ($|X| = m$), 2^{k-m} different sentences are the candidates for the invalid sentences, each corresponding to a particular subset of $R - X$. Checking the validity of each candidate sentence requires an additional scan of the relation $r(R)$. On the other hand, checking the validity of a sentence (in the top-down algorithm) requires $\mathcal{O}(n^3)$ time.

The number of sentences defined under relation schema R increases exponentially with $|R|$. This is reflected in Table 1 in the time needed for the computation of the positive border. However, the time for the computation of the positive border depends as well on the internal structure of the relation. The current implementation of the algorithms can be used for the discovery of multivalued dependencies from relations which have up to 10 attributes and include only few hundreds of tuples. Since the efficient implementation of MVD-tree is important for the performance of top-down and bottom-up algorithms, we expect that the implementation in an efficient procedural programming language could significantly improve the performance of algorithms.

5 Related work

The work on the discovery of multivalued dependencies from relations is closely related to the work on the discovery of functional dependencies from relations presented in [3, 13] and the discovery of database dependencies presented in [5]. The basic skeletons of the algorithms for the discovery of multivalued dependencies presented in Section 3 are those of algorithms for the discovery of database dependencies presented in [5]. With regards to the previous work, this paper introduces a detailed description of the top-down and bottom-up algorithms for the discovery of multivalued dependencies. In particular, the use of the dependency basis [1] for the representation of hypotheses is presented, the specialisation/generalisation relationship among the sentences is defined, the data structure for the representations of the sets of sentences called MVD-Tree is presented, and the empirical analysis of the algorithms for the discovery of multivalued dependencies is described. Finally, the procedure for the enumeration of sentences in the top-down and bottom-up algorithms is improved to avoid repeated enumeration of sentences.

The differences between the application of top-down and bottom-up algorithms for the discovery of functional and multivalued dependencies are as follows. In the case of functional dependencies, the bottom-up algorithm on average outperforms the top-down algorithm. The ratio between the computation times varies significantly (from 1:1 to 30:1 in the cases when the bottom-up algorithm performs better than the top-down algorithm [5]). As presented in Section 4.3, the opposite holds for multivalued dependencies, that is, on average the top-down algorithm outperforms the bottom-up algorithm. Again, the exact ratio depends significantly on the shape of the input relation. We will now present some reasons for the difference in performance of the top-down and bottom-up algorithms. The main reason lies in the different definitions of the multivalued and functional dependencies and, consequently, in the different complexities of the algorithms for the computation of negative cover. Secondly, the implementation of algorithms for functional dependencies does not include the efficient enumeration procedure, which could improve the performance of top-down algorithm for the functional dependencies. Thirdly, the implementation of MVD-tree in Prolog is not efficient, which significantly degrades the performance of Algorithm 4 for mvds. Finally, the inability to use the correct negative border in the case of multivalued dependencies results, among others, in poorer performance of the procedure for checking the validity of multivalued dependencies.

This work on discovery of multivalued dependencies is related to the work on discovery of interesting sentences performed by Mannila and Toivonen [8, 9]. The term 'interesting sentences' denotes the sentences of languages which can be partially ordered by a relationship which is monotonic and expresses a form of generalisation/specialisation between sentences. Such languages include association rules, functional dependencies, and multivalued dependencies. The top-down and bottom-up algorithms are closely related to the levelwise algorithm presented in [9]. The levelwise algorithm calculates the positive border of the theory using a breadth-first search algorithm which enumerates the sentences level-by-level with respect to the lattice of sentences, checks the truth of each enumerated sentence against the complete relation, and eliminates the sentences in the descendent levels which are known to be false.

Furthermore, the presented algorithms are related to the `Candidate-Elimination` algorithm introduced by Mitchell [11]. Given a set of positive and a set of negative training examples, this algorithm computes the general boundary (the set of maximally general valid sentences) G and the specific boundary (the set of maximally specific valid sentences) S. The `Candidate-Elimination` algorithm is an iterative algorithm which adjusts the borders in each step to comply with one positive or negative training example. As stated by the *Version space representation theorem*, all valid sentences which conform with the training examples are the sentences which are between G and S, that is, the sentences at least as specific as a sentence from G and at the same time at least as general as a sentence from S. The space of hypotheses has in the case of database dependencies only the general boundary G which corresponds to the previously presented positive border (i.e. S is fixed and consists of all maximally specific sentences).

Finally, attribute dependencies from database relations is related to predicate invention which is a form of constructive induction. If a functional or multivalued dependency is satisfied by a relation, the relation can be decomposed into two new relations with fewer attributes, such that the join of these new relations gives us back the original one. In logic programming terms a database relation corresponds to a predicate, and a decomposed relation corresponds to a new, intensional definition of the originally extensionally specified predicate in terms of two new extensional predicates. This approach was worked out in [4]. However, the attributes of the new relations must be already present in the original relation. [15] develops a method of function (rather than relation) decomposition which is capable of introducing new attributes.

6 Conclusions

In this paper we presented two algorithms for the computation of multivalued dependencies from relations. The hypothesis space used in the algorithms is defined in accordance with the theoretical view of the discovery of interesting sentences proposed in [9]. The top-down algorithm computes the multivalued dependencies from the input relation by using the generalisation/specialisation hierarchy of multivalued dependencies to investigate the hypotheses from the most general towards more specific hypotheses. Our bottom-up algorithm provides an alternative by first calculating the invalid dependencies, and later calculating valid dependencies from the invalid dependencies, rather than from the dataset. The implementation of the algorithms was analysed and the empirical results of the algorithms were presented.

From an application perspective, database dependencies discovery can be employed in the design and re-engineering of relational databases. The design of relational databases using functional and multivalued dependencies [6, 14] includes formal methods for the definition of relational databases which are consistent and which have minimal redundant information. Among others, multivalued dependencies serve as the basis for the definition of the *fourth normal form* [6] of relational databases which ensures that all attributes of a database relation depend, either by functional dependencies or by multivalued dependencies, exclusively on the superkeys of that relation. The re-engineering of the relational databases [7] comprises the methods for restructuring existing relational databases with respect to the various forms of database integrity constraints, including multivalued dependencies. Finally, the discovery of database dependencies can assist in the process of the discovery of other types of patterns from relational databases by revealing the internal structure of relations. This can be utilised in planning the overall process of knowledge discovery as well as for the possible restructuring of the datasets.

Acknowledgements

This work was partially supported by the Esprit Long Term Research Project 20237 (*Inductive Logic Programming 2*) and the INCO-Copernicus Network of Excellence *ILPnet2*.

References

[1] Catriel Beeri, On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases, *ACM Trans. on Database systems* 5(3), 1980.

[2] C.C. Chang & H.J. Keisler, *Model theory*, Elsevier Science Press, Third edition, 1990.

[3] P.A. Flach, Inductive characterisation of database relations, in *Proc. Fifth Int. Symposium on Methodologies for Intelligent Systems ISMIS'90*, (Z.W. Ras, M. Zemankowa, M.L. Emrich, eds.), North-Holland, Amsterdam, 1990, pp. 371-378.

[4] P.A. Flach, 'Predicate invention in Inductive Data Engineering', *Proc. European Conference on Machine Learning ECML'93*, P. Brazdil (ed.), Lecture Notes in Artificial Intelligence 667, Springer-Verlag, 1993, pp. 83-94.

[5] P.A. Flach, I. Savnik, Database dependency discovery: a machine learning approach, *AI Communications* 12(3), 1999, pp. 139-160.

[6] D. Maier, *The theory of relational databases*, Computer Science Press, Rockville, 1983.

[7] H. Mannila & K.-J. Räihä, 'The design of relational databases', Addison-Wesley, Wokingham, 1992.

[8] H. Mannila, H. Toivonen, On an algorithm for finding all interesting sentences, in *Proc. Cybernetics and Systems '96*, (R. Trappl, ed.), 1996, pp. 973-978.

[9] H. Mannila, H. Toivonen, Levelwise search and borders of theories in knowledge discovery, *Data Mining and Knowledge Discovery Journal*, 1(3), 1997, pp. 241-258.

[10] C.J. Merz, P.M. Murphy, UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html], University of California, Dept. of Information and Computer Science, 1996.

[11] T.M. Mitchell, *Machine Learning*, McGraw-Hill Series in Computer Science, 1997.

[12] I. Savnik, P.A. Flach, Program for the discovery of multivalued dependencies from relations [http://www.informatik.uni-freiburg.de/~savnik/mdep.html], Institut für informatik, Universität Freiburg, 1999.

[13] I. Savnik, P.A. Flach, Bottom-up induction of functional dependencies from relations, in *Proc. AAAI '93 Workshop on Knowledge Discovery in Databases*, (G. Piatetsky-Shapiro, ed.), 1993, pp. 174-185.

[14] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume 1, Computer Science Press, Rockville, 1988.

[15] B. Zupan, M. Bohanec, J. Demšar and I. Bratko, 'Learning by discovering concept hierarchies', *Artificial Intelligence* 109(1-2), 1999, pp. 211-242.