Unary and n-ary inclusion dependency discovery in relational databases†

Fabien De Marchi (fabien.demarchi@liris.cnrs.fr)
Université de LYON; Université LYON 1;
Laboratoire LIRIS, CNRS UMR-5205;
F-69621 Villeurbanne, France

Stéphane Lopes (stephane.lopes@prism.uvsq.fr)
Université de Versailles Saint-Quentin en Yvelines;
Laboratoire PRISM, CNRS UMR-8144;
78035 Versailles Cedex, France

Jean-Marc Petit (jean-marc.petit@liris.cnrs.fr)
Université de LYON; INSA de Lyon;
Laboratoire LIRIS, CNRS UMR-5205;
F-69621 Villeurbanne, France

Abstract. Foreign keys form one of the most fundamental constraints for relational databases. Since they are not always defined in existing databases, the discovery of foreign keys turns out to be an important and challenging task. The underlying problem is known to be the inclusion dependency (IND) inference problem.

In this paper, data-mining algorithms are devised for IND inference in a given database. We propose a two-step approach. In the first step, unary INDs are discovered thanks to a new preprocessing stage which leads to a new algorithm and to an efficient implementation. In the second step, n-ary IND inference is achieved. This step fits in the framework of levelwise algorithms used in many data-mining algorithms. Since real-world databases can suffer from some data inconsistencies, approximate INDs, i.e. INDs which almost hold, are considered. We show how they can be safely integrated into our unary and n-ary discovery algorithms.

An implementation of these algorithms has been achieved and tested against both synthetic and real-life databases. Up to our knowledge, no other algorithm does exist to solve this data-mining problem.

Keywords: Inclusion dependency discovery, relational databases

1. Introduction

Functional Dependencies (FDs) and Inclusion Dependencies (INDs) are the most famous, and most important integrity constraints in relational databases, e.g.(Casanova et al., 1984; Levene and Loizou, 1999; Mannila and Raïha, 1994). They generalize respectively keys and foreign keys which are very popular in practice. For example, INDs are necessary to express in the relational model any relationships of conceptual models

† The original publication is available at www.springerlink.com

such as ER model, and are widely used in many database applications such as database design and maintenance (Casanova et al., 1988; Levene and Vincent, 2000; Mannila and Räihä, 1986), data integration (Miller et al., 2001), or semantic query optimization (Gryz, 1998; Cheng et al., 1999). Nevertheless, considering that INDs are available in real-life databases seems to be likely optimistic: Knowledge about schema often not exist, or is lost, or does not correspond any more to the database structure. Moreover, a lot of production databases become disordered over time so that up to date information about data semantics is likely to be lost (Dasu et al., 2002).

Some recent works were proposed to discover functional dependencies (FDs) holding in a relation (Huhtala et al., 1999; Novelli and Cicchetti, 2001; Wyss et al., 2001; Lopes et al., 2002a), but IND discovery in databases has not raised great interest yet. We identify two reasons for that: 1) the difficulty of the problem due to the potential number of candidate INDs which is more than factorial in the number of attributes (cf (Casanova et al., 1984; Kantola et al., 1992) for complexity results) and 2) the fact that INDs "lack of popularity". Indeed, FDs are studied as a basic constraint of the relational model since they are used to define normal forms (e.g. BCNF or 3NF) and to define keys, very popular in practice. The same argument holds for INDs: they are also a basic constraint (Mannila and Raïha, 1994; Abiteboul et al., 1995; Levene and Loizou, 1999) and can be used to define other normal forms, such as ERNF or IDNF in order to avoid update anomalies and to ensure data coherence and integrity (see (Mannila and Raïha, 1994; Levene and Loizou, 1999; Levene and Vincent, 2000) for details on such normal forms).

Contribution In this paper, we propose algorithms to discover *all* satisfied INDs in an given database.

We propose to consider separately unary IND inference (INDs between single attributes) and n-ary IND inference (INDs between sequences of attributes). Several reasons justify this choice: unary INDs are most likely to be found in real-life databases, and at a more technical level, no sophisticated pruning can be applied when unary INDs are discovered. A characterization of satisfied unary INDs is given from a new representation of the database values, which leads to an algorithm and an efficient implementation. From discovered unary INDs, a levelwise algorithm, fitting in the framework defined in (Mannila and Toivonen, 1997), has been devised to discover n-ary INDs in a given database. We propose an Apriori-like algorithm to generate candidate INDs of size $i+1$ from satisfied INDs of size $i (i > 0)$. To deal with inconsistencies that frequently occur in databases, we consider *approximate INDs*, INDs that are almost satisfied in a database. We show how our proposal can safely be extended to approximate unary and n-ary IND discovery.

Despite the inherent complexity of this inference task, experiments on a medium-size real-life database from the web show the feasibility of this approach. We also propose a study of behavior and performance of our algorithms using medium size synthetic databases (up to 500000 tuples) available from `http://www.isima.fr/~demarchi/dbacomp/`.

This paper is an extension and a consolidation of the work published in (De Marchi et al., 2002). Approximate IND discovery has been added, together with more deepfull experiments and cost analysis for unary IND discovery. Moreover, a discussion devoted to data accesses for IND evaluation has been inserted.

Paper organization The layout of the rest of this paper is as follows: Related works is given in Section 2. Section 3 recalls some basic concepts of relational databases. Section 4 deals with IND inference: a new approach for unary IND inference is given in Section 4.1, and a levelwise algorithm is proposed to discover all remaining INDs in Section 4.2. The Section 5 extends our approach to approximate INDs. Data accesses are considered in Section 6. Experimental results on real-life and synthetic databases are presented in Section 7, and we conclude in Section 8.

2. Related works

In (Kantola et al., 1992), authors showed that it is already NP-complete to decide whether an inclusion dependency of the form $R[X] \subseteq S[Y]$ is satisfied, where X contains all the attributes of R. But authors specified that this rather negative result was obtained from a highly artificial example.

Proposals have been made to use approximations, or to find only a subset of INDs. For example in (Dasu et al., 2002), a summary of the database is computed from which a "rate of similitude" between attributes can be quickly calculated. Unary INDs can thus be found efficiently, of course with an error: some discovered unary INDs are not really satisfied, but also some satisfied unary INDs can be missed. (Lopes et al., 2002b) considers only *duplicated attributes* discovered from SQL join statements performed during a period of time over the database server. (Albrecht et al., 1995) uses natural language and name of attributes to determine candidates through a dialog tool. We proposed in (De Marchi and Petit, 2005) an approach for approximating the set of approximate satisfied INDs. The main idea is to relax the user defined treeshold, if it can lead to a more condensed set of INDs in output (as it is done in (Afrati et al., 2004) for frequent itemsets).

In (Bell and Brockhausen, 1995), exhaustive search of unary INDs is considered. Properties of INDs are exploited to reduce the number of candidate unary INDs to be tested against the database. Nevertheless, this pruning is not really effective, and a large number of tests remains to be performed against the database. The special case of unary INDs is also considered in (Bauckmann et al., 2007). As we do in this paper, authors propose to preprocess data in a form suitable for a one pass IND discovery. They experimentally show that their proposal outperforms our method; however, we argue that the two methods are very close and that the results they obtain are manly due to implementation issues, on which they give no details[1].

A particular attention to large IND discovery is given in (Koeller and Rundensteiner, 2003; De Marchi and Petit, 2003)[2]. Rather than a pure levelwise approach, these works implement "jumps" in the search space of INDs for discovering large satisfied INDs without testing an exponential number of candidates. However, they do not consider the case of unary INDs as un special case, and we argue that our approach is more appropriated in real situations, i.e. the size of satisfied INDs does not exceed five or six.

Many data mining tasks such as frequent itemsets discovery or IND discovery can fit into some common theoretical frameworks such as (Mannila and Toivonen, 1997; Calders and Wijsen, 2001). However, their contributions are theoretical only, algorithmic and implementation issues being far from authors objectives. Moreover, unary IND discovery does not fit into such frameworks and is not considered as an important sub-problem as we do in this paper.

3. Basic definitions

We briefly introduce some basic relational database concepts used in this paper (see e.g. (Mannila and Raïha, 1994; Levene and Loizou, 1999) for details).

Let R be a finite set of *attributes*. For each attribute $A \in R$, the set of all its possible values is called the *domain of* A and denoted by $Dom(A)$. A *tuple* u over R is a total mapping from R to $\bigcup_{A \in R} Dom(A)$, where $u(A) \in Dom(A), \forall A \in R$. A set r of tuples over R is called a *relation* over R, and one says that R is the *relation schema* of r. The

[1] We received no demand of our sources by authors
[2] Note that our paper was submitted before the parution of these works

cardinality of a set X is denoted by $|X|$. If $X \subseteq R$ is an attribute set[3] and u is a tuple, we denote by $u[X]$ the restriction of u to X. The projection of a relation r onto X, denoted as $\pi_X(r)$, is defined by $\pi_X(r) = \{u[X] \mid u \in r\}$.

A *database schema* $\mathbf{R}$ is a finite set of *relation schemas* R_i. A *relational database instance* $\mathbf{d}$ (or *database*) over $\mathbf{R}$ corresponds to a set of relations r_i over each R_i of $\mathbf{R}$.

An attribute sequence (e.g. $X =< A, B, C >$ or simply ABC) is an ordered set of distinct attributes. Given a sequence X, $X[i]$ refers to the ith element of the sequence. When it is clear from context, we do not distinguish a sequence from its underlying set.

Two attributes A and B are said to be *compatible* if $Dom(A) = Dom(B)$. Two distinct attribute sequences X and Y are *compatible* if $|X| = |Y| = m$ and if for $j = [1, m]$, $Dom(X[j]) = Dom(Y[j])$.

Inclusion dependencies and the notion of satisfaction of an inclusion dependency in a database are defined below.
An *inclusion dependency* (IND) over a database schema $\mathbf{R}$ is a statement of the form $R_i[X] \subseteq R_j[Y]$, where $R_i, R_j \in \mathbf{R}, X \subseteq R_i, Y \subseteq R_j$, X and Y are compatible sequences. An inclusion dependency is said to be *trivial* if it is of the form $R[X] \subseteq R[X]$. An IND $R[X] \subseteq R[Y]$ is of size i if $|X| = i$. We call *unary inclusion dependency* an IND of size 1.

Let $\mathbf{d}$ be a database over a database schema $\mathbf{R}$, where $r_i, r_j \in \mathbf{d}$ are relations over $R_i, R_j \in \mathbf{R}$ respectively. An inclusion dependency $R_i[X] \subseteq R_j[Y]$ is *satisfied* in a database $\mathbf{d}$ over $\mathbf{R}$, denoted by $\mathbf{d} \models R_i[X] \subseteq R_j[Y]$, iff $\forall u \in r_i, \exists v \in r_j$ such that $u[X] = v[Y]$ (or equivalently $\pi_X(r_i) \subseteq \pi_Y(r_j)$).

Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two sets of inclusion dependencies, $\mathcal{I}_1$ is a *cover* of $\mathcal{I}_2$ if $\mathcal{I}_1 \models \mathcal{I}_2$ (this notation means that each dependency in $\mathcal{I}_2$ holds in any database satisfying all the dependencies in $\mathcal{I}_1$) and $\mathcal{I}_2 \models \mathcal{I}_1$.

A sound and complete axiomatization for INDs is made up of three inference rules (Casanova et al., 1984):

1. (reflexivity) $R[A_1, ..., A_n] \subseteq R[A_1, ..., A_n]$

2. (projection and permutation)
   $if R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n]$ then
   $R[A_{\sigma 1}, ..., A_{\sigma m}] \subseteq S[B_{\sigma 1}, ..., B_{\sigma m}]$ for each sequence $\sigma 1, ..., \sigma m$ of distinct integers from $\{1, ..., n\}$

3. (transitivity) if $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n]$ and
   $S[B_1, ..., B_n] \subseteq T[C_1, ..., C_n]$ then $R[A_1, ..., A_n] \subseteq T[C_1, ..., C_n]$

[3] Letters from the beginning of the alphabet introduce single attributes whereas letters from the end introduce attribute sets.

4. Inclusion dependency inference

The IND inference problem can be formulated as follows: "Given a database $\mathbf{d}$ over a database schema $\mathbf{R}$, find a cover of all non trivial inclusion dependencies $R[X] \subseteq S[Y]$, $R, S \in R$, such that $\mathbf{d} \models R[X] \subseteq S[Y]$".

In this paper, we propose to re-formulate this problem into two sub-problems: the former is the unary IND inference problem, and the latter is the n-ary IND inference problem being understood that unary INDs have been discovered.

4.1. Unary inclusion dependency inference

We propose a new and efficient algorithm to discover unary INDs satisfied in a given database. The idea is to build a binary relation which associates each values of the database with attributes having this value. Unary INDs can then be efficiently computed from this binary relation. We give the details below.

4.1.1. *Data preprocessing*

Given a database $\mathbf{d}$ over a database schema $\mathbf{R}$, and a data type t of $\mathbf{d}$, we define an *extraction context*[4] denoted by the triple $\mathbb{D}_t(\mathbf{d}) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$ as follows:

- $\mathbb{U} = \{R.A \mid \text{ A is of type t }, A \in R, R \in \mathbf{R}\}$. $\mathbb{U}$ is the set of attributes[5] whose type is t;
- $\mathbb{V} = \{v \in \pi_A(r) \mid R.A \in \mathbb{U}, r \in \mathbf{d}, r \text{ defined over } R\}$. $\mathbb{V}$ is the set of values taken by attributes in their relations;
- $\mathbb{B} \subseteq \mathbb{V} \times \mathbb{U}$ is a binary relation defined by:
  $(v, R.A) \in \mathbb{B} \Longleftrightarrow v \in \pi_A(r)$, where $r \in \mathbf{d}$ and r defined over R.

Example 1 *Let us consider the database* $\mathbf{d}$ *given in Table I as a running example. Domains of attributes of these relations are of three types:* int, real, string. *For the type* int, $\mathbb{U} = \{A, C, E, G, K\}$ *and* $\mathbb{V} = \{1, 2, 3, 4, 6, 7, 9\}$. *For instance, the value 1 appears in* $\pi_A(r), \pi_E(s)$ *and* $\pi_K(t)$, *and thus* $(1, A), (1, E)$ *and* $(1, K) \in \mathbb{B}$.
Table II summarizes the extraction context associated with int, real *and* string.

[4] This name comes from Formal Concept Analysis (Ganter and Wille, 1999)
[5] When clear from context, we will omit to prefix attributes by their relation schema.

Table I. A running example

r

A	B	C	D
1	X	3	11.0
1	X	3	12.0
2	Y	4	11.0
1	X	3	13.0

s

E	F	G	H
1	X	3	11.0
2	Y	4	12.0
4	Z	6	14.0
7	W	9	14.0

t

I	J	K	L
11.0	11.0	1	X
12.0	12.0	2	Y
11.0	14.0	4	Z
11.0	9.0	7	W
13.0	13.0	9	R

Table II. Extraction contexts associated with the database **d**.

int

$\mathbb{V}$	$\mathbb{U}$
1	A E K
2	A E K
3	C G
4	C E G K
6	G
7	E K
9	G K

real

$\mathbb{V}$	$\mathbb{U}$
9.0	J
11.0	D H I J
12.0	D H I J
13.0	D I J
14.0	H J

string

$\mathbb{V}$	$\mathbb{U}$
R	L
X	B F L
Y	B F L
Z	F L
W	F L

4.1.2. *A new algorithm for unary IND inference*
With this new data organization, an interesting characterization of unary INDs can be given. Informally, if the IND $A \subseteq B$ is satisfied, then by construction for every value v such that (v, A) belongs to the extraction context, we have (v, B) belongs to the extraction context.

Proposition 1 *Given a database* **d***, a data type* t *and an extraction context* $\mathbb{D}_t(\boldsymbol{d}) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$,

$$\boldsymbol{d} \models A \subseteq B \Longleftrightarrow B \in \bigcap_{v \in \mathbb{V} | (v,A) \in \mathbb{B}} \{C \in \mathbb{U} \mid (v, C) \in \mathbb{B}\}$$

where $A, B \in \mathbb{U}$. Proof *Let* $A \in R$, $B \in S$ *such that* $\boldsymbol{d} \models A \subseteq B$.

$\Longleftrightarrow \forall v \in \pi_A(r), \exists u \in s$ *such that* $u[B] = v$
$\Longleftrightarrow \forall v \in \mathbb{V}$ *such that* $(v, A) \in \mathbb{B}$*, we have* $(v, B) \in \mathbb{B}$ $\square$

In fact, an extraction context can be seen as a transaction database, in which attributes are items and values are transactions. Unary INDs correspond to exact association rules (i.e. association rules whose confidence is 100 %) whose left and right-hand sides are made up of only one attribute in this transaction database.

Thus, the whole task of unary IND inference can be done in only *one pass* of each extraction context. Algorithm 1 follows from Proposition 1 which gives its correctness. It finds all unary INDs in a database **d**, between attributes defined on a type t, taking in input the extraction context as described before. For every attribute A, we denote by $rhs(A)$ (for right-hand side) the set of attributes B such that $A \subseteq B$.

Algorithm 1 Unary IND inference

Input: the extraction context $(\mathbb{V}, \mathbb{U}, \mathbb{B})$, associated with **d** and t.
Output: $\mathcal{I}_1$ the set of unary INDs satisfied by **d** between attributes of type t.
```
1: for all A ∈ U do rhs(A) = U;
2: for all v ∈ V do
3:     for all A s.t. (v, A) ∈ B do
4:         rhs(A) = rhs(A) ∩ {B | (v, B) ∈ B};
5: for all A ∈ U do
6:     for all B ∈ rhs(A) \ {A} do
7:         I1 = I1 ∪ {A ⊆ B};
8: return I1.
```

This algorithm is linear with respect to the size of the binary relation of the extraction context. Indeed, the intersection of two sorted sets is equal to two times the size of the smallest set. Here, the size of $rhs(A)$ is smaller than the size of $\mathbb{U}$ by several order of magnitude. Thus the complexity of intersection can be assimilate as a constant.

Example 2 *Let us consider the type* `int` *(cf Table II) in example 1. The initialization step (line 1) gives:* $rhs(A) = rhs(C) = \ldots = rhs(K) = \{A, C, E, G, K\}$.
Then, we consider the set of attributes in the first line of the extraction context: $l_1 = \{A, E, K\}$. *For each attribute in* l_1, *its* rhs *set is updated (line 4) as follows:*

- $rhs(A) = \{A, E, K\}$
- $rhs(E) = \{A, E, K\}$
- $rhs(K) = \{A, E, K\}$
- $rhs(C) = \{A, C, E, G, K\}$ *(unchanged)*
- $rhs(G) = \{A, C, E, G, K\}$ *(unchanged)*

These operations are repeated for each value of the extraction context (line 2). Finally, after one pass, the result is:

- $rhs(A) = \{A, E, K\}$
- $rhs(E) = \{E, K\}$
- $rhs(K) = \{K\}$
- $rhs(C) = \{C, G\}$
- $rhs(G) = \{G\}$

From these sets, unary INDs between attributes of type `int` *are (lines 5,6 and 7):* $\{A \subseteq E, A \subseteq K, C \subseteq G, E \subseteq K\}$.
The same operation has to be repeated for each data type, and then, thanks to property 1, we deduce the following set of unary inclusion dependencies satisfied by $\mathbf{d}$: $\{A \subseteq E, A \subseteq K, B \subseteq F, B \subseteq L, C \subseteq G, D \subseteq I, D \subseteq J, E \subseteq K, F \subseteq L, H \subseteq J, I \subseteq D, I \subseteq J\}$.

4.2. A levelwise algorithm for n-ary IND inference

Once unary INDs are known, the problem we are interested in can be re-formulated as follows: "Given a database $\mathbf{d}$ over a database schema $\mathbf{R}$ and the set of unary INDs verified by $\mathbf{d}$, find a cover of all non trivial inclusion dependencies $R[X] \subseteq S[Y]$, $R, S \in R$, such that $\mathbf{d} \models R[X] \subseteq S[Y]$".

We first recall how IND properties justify a levelwise approach to achieve their inference (Mannila and Toivonen, 1997). Then, we give an algorithm, with a natural method to generate candidate INDs of size $i+1$ from satisfied INDs of size i.

4.2.1. *Definition of the search space*
Candidate INDs are composed of a left-hand side and a right-hand side. Given a set of attributes, we do not have to consider all the permutations to build a left-hand side or a right-hand side, thanks to the second inference rule presented in section 3.

Example 3 *Let $R[AB] \subseteq S[EF]$ and $R[AB] \subseteq T[KL]$ be two satisfied INDs. Then, thanks to the second inference rule of INDs (permutation), $R[BA] \subseteq S[FE]$ and $R[BA] \subseteq T[LK]$ are also satisfied.*

Then, we are faced with the following problem: in which order attribute sequences have to be built to avoid considering several permutations of the same IND ? Therefore, we have to set an order for attributes of the left-hand sides or for attributes of the right-hand sides. The most natural one has been chosen, i.e. we choose the lexicographic order for attributes of the left-hand sides.

4.2.2. *Reduction of the search space*
Between two candidate INDs, a specialization relation $\preceq$ can be defined as follows:

Let $I_1 : R_i[X] \subseteq R_j[Y]$ and $I_2 : R'_i[X'] \subseteq R'_j[Y']$ be two candidate INDs. We define $I_2 \preceq I_1$ iff:

- $R_i = R'_i$ and $R_j = R'_j$ and

- $X' =< A_1, ..., A_k >$, $Y' =< B_1, ..., B_k >$, and there exists a set of indices $i_1 < ... < i_h \in \{1, ..., k\}$ with $h \leq k$ such that $X =< A_{i_1}, ..., A_{i_h} >$, $Y =< B_{i_1}, ..., B_{i_h} >$[6].

Note that X, Y, X' and Y' are sequences, and thus the specialization relation respects the order of attributes.

Example 4 *We have $(R_i[AC] \subseteq R_j[EG]) \preceq (R_i[ABC] \subseteq R_j[EFG])$, but $(R_i[AC] \subseteq R_j[GE]) \not\preceq (R_i[ABC] \subseteq R_j[EFG])$.*

We note $I_1 \prec I_2$ if $I_1 \preceq I_2$ and $I_2 \not\preceq I_1$.

From the second inference rule of INDs, we can deduce the following property, which justifies a levelwise approach for IND inference.

Property 1 *Let I_1, I_2 be 2 candidate INDs such that $I_1 \preceq I_2$.*
If $\mathbf{d} \not\models I_1$ then $\mathbf{d} \not\models I_2$.

This property extends the a-priori property to our problem; we say that the relation $\preceq$ is anti-monotone ((Han and Kamber, 2000)) w.r.t. the satisfiability of INDs. Then, knowing unsatisfied INDs at a given

[6] This definition is slightly different from that given in (Mannila and Toivonen, 1997). Here, we impose an order for index $i_1, ..., i_h$ without any loss of information.

level, allows us to prune candidates for the next level. More precisely, only satisfied INDs will be used to generate candidate INDs for the next level. Thus, the search space will be considerably reduced, for levels higher than one.

4.2.3. *The algorithm*

From now, notations given in Table III will be used throughout the paper.

Table III. Notations

$\mathcal{C}_i$	Set of candidate inclusion dependencies of size i.
$\mathcal{I}_i$	Set of satisfied inclusion dependencies of size i.
$I.lhs$	Left-hand side sequence of the IND I
$I.rhs$	Right-hand side sequence of the IND I
$X.rel$	Relation schema of attributes of the sequence X

Algorithm 2 finds a cover of INDs holding in a given database $\mathbf{d}$, taking in input the set of unary INDs satisfied by $\mathbf{d}$ (cf section 4.1). The first step consists in computing candidate INDs of size 2, from satisfied INDs of size 1. Then, these candidates are tested against the database. From the satisfied ones, candidate INDs of size 3 are generated and then tested against the database. This process is repeated until no more candidates can be computed.

Algorithm 2 MIND

Input: $\mathbf{d}$ a database, and $\mathcal{I}_1$ the set of unary INDs satisfied by $\mathbf{d}$.
Output: Inclusion dependencies satisfied by $\mathbf{d}$

$\mathcal{C}_2 := GenNext(\mathcal{I}_1)$;
$i := 2$;
while $\mathcal{C}_i \neq \emptyset$ **do**
  forall $I \in \mathcal{C}_i$ **do**
    if $\mathbf{d} \models I$ **then**
      $\mathcal{I}_i := \mathcal{I}_i \cup \{I\}$;
  $\mathcal{C}_{i+1} := GenNext(\mathcal{I}_i)$;
  $i := i + 1$;
end while
return $\cup_{j<i}\mathcal{I}_j$

The theoretical complexity of such an algorithm is equal to the cost of one test against the database (here the test of an IND), times the number of satisfied INDs plus the number of not satisfied INDs whose all specializations are satisfied, i.e. the so-called *negative border* of satisfied

INDs. For other complexity results of levelwise algorithms, the reader is referred to (Mannila and Toivonen, 1997).

4.2.4. *Candidate INDs generation*

The function *GenNext* extends the principle of candidate generation, whose archetype is the AprioriGen function (Agrawal and Srikant, 1994) used for frequent itemsets discovery. Algorithm 3 details the function *GenNext*. It is made up of two main parts: a generation step and a pruning step, both based on the anti-monotony property of the relation $\preceq$ w.r.t. INDs satisfiability (cf. property 1).

Algorithm 3 *GenNext* : Generation of candidate INDs of size $i+1$

Input: $\mathcal{I}_i$, inclusion dependencies of size i.
Output: $\mathcal{C}_{i+1}$, sequence of candidate inclusion dependencies of size i+1

1: **insert into** $\mathcal{C}_{i+1}$
2: **select** $p.lhs.rel$"["$p.lhs[1], p.lhs[2], ..., p.lhs[i], q.lhs[i]$"]" " $\subseteq$ " $p.rhs.rel$"["$p.rhs[1], p.rhs[2], ..., p.rhs[i], q.rhs[i]$"]"
3: **from** $\mathcal{I}_i$ p, $\mathcal{I}_i$ q
4: **where** $p.lhs.rel = q.lhs.rel$ **and** $p.rhs.rel = q.rhs.rel$
5:     **and** $p.lhs[1] = q.lhs[1]$ **and** $p.rhs[1] = q.rhs[1]$
6:     **and** ...
7:     **and** $p.lhs[i-1] = q.lhs[i-1]$ **and** $p.rhs[i-1] = q.rhs[i-1]$
8:     **and** $p.lhs[i] < q.lhs[i]$
9:     **and** $p.rhs[i] <> q.rhs[i]$
10: **for all** $I \in \mathcal{C}_{i+1}$ **do**
11:   **for all** $J \prec I$ and J of size i **do**
12:     **if** $J \notin \mathcal{I}_i$ **then**
13:       $\mathcal{C}_{i+1} = \mathcal{C}_{i+1} \setminus \{I\}$
14:     **end if**
15:   **end for**
16: **end for**

The generation step (lines 1 to 8) constructs candidate INDs from satisfied INDs of the previous level. Let $I_1 = R_i[XA] \subseteq R_j[YC]$ and $I_2 = R_i[XB] \subseteq R_j[YD]$ be two satisfied INDs of level i, with $|X| = |Y| = i-1$, $A < B$ and $C \neq D$. Then the candidate: $I_3 = R_i[XAB] \subseteq R_j[YCD]$ is formed.

Example 5
From the running example, Table IV shows satisfied INDs at level 1 ($\mathcal{I}_1$) in the first column, classified by relations. Candidate INDs of size 2 are represented in the second column of the table.

Table IV. Generation of $\mathcal{C}_2$ from $\mathcal{I}_1$.

	$\mathcal{I}_1$	$\mathcal{C}_2$
R to S	$R[A] \subseteq S[E]$ $R[B] \subseteq S[F]$ $R[C] \subseteq S[G]$	$R[AB] \subseteq S[EF]$ $R[AC] \subseteq S[EG]$ $R[BC] \subseteq S[FG]$
R to T	$R[A] \subseteq T[K]$ $R[B] \subseteq T[L]$ $R[D] \subseteq T[I]$ $R[D] \subseteq T[J]$	$R[AB] \subseteq T[KL]$ $R[AD] \subseteq T[KI]$ $R[AD] \subseteq T[KJ]$ $R[BD] \subseteq T[LI]$ $R[BD] \subseteq T[LJ]$
S to T	$S[E] \subseteq T[K]$ $S[F] \subseteq T[L]$ $S[H] \subseteq T[J]$	$S[EF] \subseteq T[KL]$ $S[EH] \subseteq T[KJ]$ $S[FH] \subseteq T[LJ]$
T to R	$T[I] \subseteq R[D]$	
T to T	$T[I] \subseteq T[J]$	

The pruning step (lines 9 to 14) removes all candidates which do not comply with the anti-monotony property.

Example 6 *To illustrate this pruning step, suppose that we have only two INDs satisfied at the level 2: $R[AB] \subseteq S[EF]$ and $R[AC] \subseteq S[EG]$. Then, the generation step constructs $R[ABC] \subseteq S[EFG]$ as a candidate IND at the level 3.*
The pruning step verifies that each IND of size 2 which specializes $R[ABC] \subseteq S[EFG]$ are satisfied at level 2. Since $R[BC] \subseteq S[FG]$ is not satisfied, the candidate is removed from $\mathcal{C}_2$.

Remark on GenNext The *GenNext* algorithm generates INDs without repeated attributes neither in the left-hand sides nor in the right-hand sides. However, repeated attributes are sometimes allowed in IND expressions, see for instance (Mitchell, 1983), and could be useful in some applications. We now point out how they could be safely integrated into *GenNext*:

- in line 8, replace the operator $<$ by $\leq$ to allow repeated attributes in the left-hand sides;
- remove line 9 to generate repeated attributes in the right-hand sides;

5. Approximate IND discovery

The algorithms presented so far deals with the discovery of inclusion dependencies satisfied in a database. When some data inconsistencies exist in the database, INDs that should be valid in the database schema might not be satisfied in that particular instance. As a consequence, we need to define some error measures for the satisfaction of an IND, in such a way that small inconsistencies can be taken into account, as already done for Functional Dependencies (FDs) (Kivinen and Mannila, 1995). In fact, this notion is known as *approximate IND* and an interesting error measure is given by function g'_3 (the name g'_3 comes from the measure g_3 for FDs (Kivinen and Mannila, 1995)) and is defined as follows (Lopes et al., 2002b):

$$g'_3(R[X] \subseteq S[Y], \mathbf{d}) =$$

$$1 - \frac{max\{|\pi_X(r')| \text{ s.t. } r' \subseteq r \text{ and } (\mathbf{d} - \{r\}) \cup \{r'\} \models R[X] \subseteq S[Y]\}}{|\pi_X(r)|}$$

Informally, g'_3 is the proportion of values in X one has to remove (independently of their occurrences) to obtain a database $\mathbf{d'}$ such that $\mathbf{d'} \models R[X] \subseteq S[Y]$.

Definition 1 *Given an user-supplied threshold $\epsilon \in [0;1]$, an approximate IND I is satisfied in a database $\mathbf{d}$ with respect to ϵ, denoted by $\mathbf{d} \models_\epsilon I$, iff $g'_3(I, \mathbf{d}) \leq \epsilon$.*

Example 7 *In Table I, $g'_3(S[H] \subseteq T[I], \boldsymbol{d}) = 1/3$. If we consider a threshold $\epsilon = 0.33$, then $\boldsymbol{d} \models_{0.33} S[H] \subseteq T[I]$.*

An algebraic property follows from the definition of g'_3, since it is just enough to compute the proportion of values of the left-hand side which do not belong to the right-hand side.

$$g'_3(R[X] \subseteq S[Y], \mathbf{d}) = \frac{|\pi_X(r) - \pi_Y(s)|}{|\pi_X(r)|}$$

SQL queries can thus quite easily be devised to compute that error measure.

5.1. Unary approximate IND discovery

Approximate unary INDs can be discovered through the same organization of data as for exact unary IND discovery. Once the extraction context has been computed (cf section 4.1), Proposition 2 gives a characterization of approximate unary INDs.

Proposition 2 *Given an user-supplied threshold ϵ, a database $\boldsymbol{d}$, a data type t and the corresponding extraction context $\mathbb{D}_t(\boldsymbol{d}) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$,*

$$\boldsymbol{d} \models_\epsilon A \subseteq B$$

$$\Longleftrightarrow$$

$$1 - \sum_{v \in \mathbb{V} | (v,A) \in \mathbb{B}} \frac{|\{v \in \mathbb{V} \mid (v,A) \in \mathbb{B} \text{ and } (v,B) \in \mathbb{B}\}|}{|\{v \in \mathbb{V} \mid (v,A) \in \mathbb{B}\}|} \leq \epsilon$$

where $A, B \in \mathbb{U}$.

Thanks to this characterization, satisfied approximate INDs can be discovered in only one pass of the extraction context. Note that, if we see the extraction context as a transaction database, in which attributes are items and values are transactions, satisfied approximate unary INDs with respect to a threshold ϵ correspond to *approximate association rules* whose left and right-hand sides are made up of only one attribute, with a confidence higher than $(1 - \epsilon)$.

Algorithm 4 computes approximate unary INDs in a database, for a given type t and a threshold ϵ. The idea is quite similar of that used for exact unary INDs. For each attribute A, we construct a set $rhs(A) = \{< B, n_{AB} >\}$ where B denotes attributes of the same type as A, and n_{AB} is the number of lines in the extraction context where B and A appear together.

Algorithm 4 Approximate unary IND inference

Input: the extraction context $(\mathbb{V}, \mathbb{U}, \mathbb{B})$, associated with $\mathbf{d}$ and t, a threshold ϵ.
Output: $\mathcal{AI}_1$ the set of approximate unary INDs satisfied by $\mathbf{d}$ between attributes of type t.

1: **for all** $A \in \mathbb{U}$ **do** $rhs(A) = \{< B, 0 >\mid B \in \mathbb{U}\}$;
2: **for all** $v \in \mathbb{V}$ **do**
3:     **for all** A s.t. $(v, A) \in \mathbb{B}$ **do**
4:         **for all** $< B, n_{AB} > \in rhs(A)$ s.t. $(v, B) \in \mathbb{B}$ **do**
5:             $n_{AB} = n_{AB} + 1$;
6: **for all** $A \in \mathbb{U}$ **do**
7:     **for all** $< B, n_{AB} > \in rhs(A) \setminus \{< A, n_{AA} >\}$ **do**
8:         **if** $1 - (n_{AB}/n_{AA}) \leq \epsilon$ **then**
9:             $\mathcal{AI}_1 = \mathcal{AI}_1 \cup \{A \subseteq B\}$;
10: return $\mathcal{AI}_1$.

The complexity of this algorithm is similar to the complexity of Algorithm 1, i.e. linear in the size of the binary relation of the extraction context. Indeed, the size of $rhs(A)$ being very small with respect to the size of $\mathbb{U}$, the complexity of lines 4 and 5 can be assimilate to a constant. The only overhead to compute approximate unary INDs with respect to exact unary INDs just concerns the size of extraction contexts, to stock the number of co-occurrences of attributes.

The next example illustrates approximate unary IND inference on the running example.

Example 8 *Let us consider the type* real *(cf Table II) given in example 1. An initialization step constructs the following sets:*
$rhs(D) = rhs(H) = rhs(I) = rhs(J) =$
$\{< D, 0 >, < H, 0 >, < I, 0 >, < J, 0 >\}$.
Then, consider the first line of the extraction context: $l_1 = \{J\}$. *The* rhs *sets are updated as follows:*
$rhs(D) = \{< D, 0 >, < H, 0 >, < I, 0 >, < J, 0 >\}$,
$rhs(H) = \{< D, 0 >, < H, 0 >, < I, 0 >, < J, 0 >\}$,
$rhs(I) = \{< D, 0 >, < H, 0 >, < I, 0 >, < J, 0 >\}$,
$rhs(J) = \{< D, 0 >, < H, 0 >, < I, 0 >, < J, 1 >\}$.
From the second line $l_2 = \{D, H, I, J\}$ *we get:*
$rhs(D) = \{< D, 1 >, < H, 1 >, < I, 1 >, < J, 1 >\}$,
$rhs(H) = \{< D, 1 >, < H, 1 >, < I, 1 >, < J, 1 >\}$,
$rhs(I) = \{< D, 1 >, < H, 1 >, < I, 1 >, < J, 1 >\}$,
$rhs(J) = \{< D, 1 >, < H, 1 >, < I, 1 >, < J, 2 >\}$.
These operations are repeated for each line of the extraction context. Finally, after one pass, the result is:
$rhs(D) = \{< D, 3 >, < H, 2 >, < I, 3 >, < J, 3 >\}$,
$rhs(H) = \{< D, 2 >, < H, 3 >, < I, 2 >, < J, 3 >\}$,
$rhs(I) = \{< D, 3 >, < H, 2 >, < I, 3 >, < J, 3 >\}$,
$rhs(J) = \{< D, 3 >, < H, 3 >, < I, 3 >, < J, 5 >\}$.
From these sets, the error measure g_3' *can be computed for each unary IND. For example:* $g_3'(D \subseteq H, \boldsymbol{d}) = 1 - \frac{2}{3}$, $g_3'(J \subseteq I, \boldsymbol{d}) = 1 - \frac{3}{5}$.
The output is made up of all INDs with an error measure lower than or equal to the threshold.

5.2. N-ary approximate IND discovery

For approximate INDs of size 2 or more, a levelwise approach is still well founded.

Property 2 *Let* I_1, I_2 *be 2 candidate INDs such that* $I_1 \preceq I_2$.
If $\boldsymbol{d} \not\models_\epsilon I_1$ *then* $\boldsymbol{d} \not\models_\epsilon I_2$.

Proof *This result comes from the consideration that, given* I_1 *and* I_2 *two candidate INDs in a database* $\mathbf{d}$*, we have:* $I_1 \preceq I_2 \Rightarrow g_3'(I_1, \mathbf{d}) \leq g_3'(I_2, \mathbf{d})$*, which is obvious from the definition of* g_3' *and of the relation* $\preceq$. □

Example 9
In the running example of Table I, we have $g_3'(S[H] \subseteq T[I], \boldsymbol{d}) = 1/3$*, and* $g_3'(S[G] \subseteq T[K], \boldsymbol{d}) = 1/2$. *Thanks to property 2, we know that*

$g_3'(S[GH] \subseteq T[KI], \boldsymbol{d}) \geq 1/2$. *Here we have* $g_3'(S[GH] \subseteq T[KI], \boldsymbol{d}) = 1$.

Thus, Algorithm 2 can be applied with only minor changes: adding an input threshold ϵ, and replacing $\mathbf{d} \models I$ by $\mathbf{d} \models_\epsilon I$ (line 5).

6. How to access the database ?

Whatever the algorithm used to infer INDs, particular care should be devoted to test the satisfaction of an IND. SQL queries can be devised to achieve this task, according to the simple algebraic property given below: Let $\mathbf{d}$ be a database over a database schema $\mathbf{R}$, where $r, s \in \mathbf{d}$ are relations over $R, S \in \mathbf{R}$ respectively. We have:

$$\mathbf{d} \models R[X] \subseteq S[Y] \Longleftrightarrow |\pi_X(r) - \pi_Y(s)| = 0$$

This property can be efficiently tested for instance with a *NOT EXISTS* query.
Given $R[A_1...A_n] \subseteq S[B_1...B_n]$, the Figure 1 shows SQL query implementing the satisfaction test against a database (under Oracle v9i).

```
SELECT /*+ USE_HASH (r) */ X
FROM r
WHERE NOT EXISTS ( SELECT *
                    FROM s
                    WHERE r.A_1 = s.B_1
                    AND ...
                    AND r.A_n = s.B_n  )
AND ROWNUM < 2
```

Figure 1. NOT EXISTS query to test $R[A_1...A_n] \subseteq S[B_1...B_n]$

This query enforces the optimizer to implement the join using hash tables (hint *USE_HASH*). Among all the tested physical plans, this query gives the best results. The selection condition `ROWNUM<2` enforces the query to stop as soon as a tuple contradicts the IND. This technique implies better performances in case of false INDs.

We have also compare the use of a *NOT EXISTS* query with the use of *NOT IN* query and *MINUS* query, which are also natural to test IND satisfaction. In our experimental conditions, execution times using *NOT EXISTS* query were significantly better than for other queries, in the both cases of satisfied and unsatisfied INDs. We do not give details here about these comparisons, since their implementation, optimization and performances strongly depend on the RDBMS.

Another solution would be to use a *loosely coupled approach* (e.g. PL/SQL programs) to test an IND against a database. Despite many optimizations performed on PL/SQL code under Oracle to avoid so called *context switching* (Sarawagi et al., 2000) between the RDBMS and the external process, performances of such an approach were poor in our experimental conditions. We are currently working on *tightly coupled approach* using User Defined Functions of DB2 RDBMS as proposed in (Sarawagi et al., 2000).

For testing approximate n-ary INDs, the query of Figure 1 can be easily adapted to compute the number of distinct values which disqualify the IND. However, to compute the error measure, it is necessary to know the left-hand side cardinality of the IND, for instance thanks to a `COUNT DISTINCT` query.

Since the $NOT\ EXISTS$ queries gave us better results in our experimental conditions, they were used in our experiments described in the next section.

7. Experimental results

7.1. Materials

7.1.0.1. *Experimental conditions* All our experiments were performed on an Intel Pentium III with a CPU clock rate of 500 Mhz, 384 MB of main memory and running Windows 2000 professional. Algorithms were implemented using the C++ language and STL (Standard Template Library). Oracle 9 was used to perform tests while RDBMS accesses were done via ODBC.

7.1.0.2. *Test databases* The test databases are of two kind: a real-life database and a set of synthetic databases

The real-life database is the *movies* database, available from the web at the UCI KDD Archive (Bay, 1999). It contains information about a list of movies; details are given in Table V.

The database has been imported in Oracle 9 using SQL*Loader. Schema information is available on the web site; the set of valid INDs is made up of 2 INDs of size 3, 7 INDs of size 2 and 8 INDs of size 1. We will reuse this information to assess the quality of the knowledge we will able to produce from the data.

To test performances of our programs, we created synthetic databases, *all with the same set of satisfied INDs.* In order to be as close as possible from real-life databases, we chose to enforce the following INDs in our synthetic databases: 10 unary INDs, 15 INDs of size 2, 20 of size

Table V. Characteristics of the *movies* database

relations	Nb of attributes	Nb of rows
ACTORS	12	6728
CASTS	7	45442
MOVIES	11	11405
PEOPLE	11	3304
REMAKES	7	1188
STUDIOS	9	195

3, 15 of size 4, 6 of size 5 and 1 of size 6. To do so, we have implemented a *chase*-like procedure to fill in these databases. All attributes in these databases are of the same type and have approximately 70% of distinct values. The databases only differ in the number of rows and the number of attributes, which can be set to study the scalability of our proposal. The synthetic databases can be downloaded from `http://www.isima.fr/~demarchi/dbacomp/`.

7.2. Unary IND discovery

We ran our algorithm for unary IND discovery against the *movies* database, and we found 13 unary INDs. The 8 unary INDs valid in the schema of *movies* were discovered, and 5 unary INDs were "accidentally" satisfied in this instance, i.e. they are not property of the schema. The execution time for unary IND inference against *movies* was only of 12 seconds.

In order to show the efficiency of the data re-organization, we propose to compare our data-mining algorithm with the approach consisting in generating candidate unary INDs and then test them against the database (Bell and Brockhausen, 1995) referred to as *test and generate approach* in the sequel. In this case, if N is the number of attributes (all of the same data type), approximately N^2 candidates[7] have to be tested against the database. If the projection cardinality of each attribute is known a priori, only $N^2/2$ candidates have to be tested, since $R[A] \subseteq S[B] \Rightarrow |\pi_A(r)| \leq |\pi_B(r)|$. In the sequel, we consider that the test and generate approach consists in testing $N^2/2$ candidates.

Initially, we set the number of attributes to 50, and test the two methods for different numbers of tuples (Figure 2). Then, we set the

[7] Since a very small proportion of these candidates are satisfied, few of them can be avoided using transitivity property. Without loss of generality, we do not consider such optimization.

Figure 2. Execution time to discover unary INDs for different numbers of rows and 50 attributes

Figure 3. Execution time to discover unary INDs for different numbers of attributes and 90 000 rows

number of tuples to 90 000, while varying the number of attributes (Figure 3).

7.2.1. *Discussion*

Results in Figures 2 and 3 show with evidence that our algorithm outperforms the test and generate approach.

Number of scans The benefit of our algorithm can be explained by studying the number of scans.

Let us consider without loss of generality a database made up of N attributes of the same type, all relations with n tuples. As mentioned, the test and generate approach leads to approximately $N^2/2$ tests. If each test is performed through a SQL query, it leads to one full scan of each side of the candidates, i.e. $2.(N^2/2) = N^2$ full scans.

In our approach, to build the extraction contexts, attributes are first scanned by the RDBMS. Then, each attribute is given, through cursors, to an external program which inserts the values in the extraction context in construction. Thus, there are N full scans (on tables) provided by the

RDBMS, and again N full scans (on cursors) to insert in the extraction context, i.e. $2.N$ full scans.

If extraction context can fit in main memory, our approach does not incur I/O cost anymore. Thus, it leads to reduce the number of scans by $N/2$. In Figure 2, one can verify that there is a factor of $N/2$ (here 50/2=25) between the two methods for 90000 tuples.

In Figure 3, the number of tuples is fixed, and thus the time complexity for a scan is constant. Note that the time complexity for the test and generate approach is effectively quadratic in the number of attributes while it is linear for our data-mining algorithm.

Size of the extraction context For N attributes of a given type, the size of the extraction context clearly depends on the number of distinct values token by these attributes. If n_t is the number of distinct values of type t, and c_t is the number of bits needed to store a value of type t, then the size of the extraction context corresponding to t is approximatively $(N + c_t).n_t$ bits. In our largest synthetic database, there are 50 attributes of the same type and 350.000 distinct integer values (32 bits) in the database, that leads to a size of 3,5 Megabytes. For a large production database, with 500 attributes and 3.500.000 distinct values, the extraction context would have a size of 350 Megabytes, that can be kept in main memory.

7.3. N-ary IND discovery

We ran our program for n-ary IND discovery against movies, and we found the following INDs: 1 IND of size 3 and 6 INDs of size 2. One of the INDs of size 2 was not property of the *movies* schema. Remark that 1 IND of size 3 and 2 INDs of size 2 were not discovered by the program, due to some errors occurring in *movies*. The execution time was only 15 seconds for the whole task of IND discovery (unary + n-ary IND discovery).

Figure 4 gives execution times against synthetic databases for the whole task of IND inference, including times to find unary INDs.

Despite the inherent complexity of the IND inference problem, execution times do not exceed 50 minutes in our experimental conditions. The execution time is approximately linear w.r.t. the number of tuple, which comes from the fact that execution time for a test is also linear w.r.t the number of tuples.

Figure 4. Execution time for an exhaustive IND discovery in synthetic databases

7.4. Approximate IND discovery

Since our program for n-ary IND discovery missed some INDs that were valid in the schema of *movies*, the program has been run again to discover approximate INDs, allowing an error rate of 0.03. This time, 2 INDs of size 3, 9 INDs of size 2 and 13 INDs of size 1 were discovered. All the INDs valid in the schema were among the discovered knowledge; 2 INDs of size 2 and 5 INDs of size 1 were not properties of the *movies* schema. The execution time for this task was only of 21 seconds.

8. Conclusion

The discovery of inclusion dependencies in relational databases is a (relatively new) data mining problem useful in many database applications. To the best of our knowledge, the work presented in this paper is the first one addressing the different aspects of this important problem: we first considered the problem of unary IND discovery, since they turn out to be very common in operational databases. We proposed an approach whose interest is to reduce the number of database accesses w.r.t. naive approaches. For the sake of completeness, we proposed to discover all remaining INDs using the framework of levelwise algorithms. We also showed how approximate INDs could be easily discovered with our approach; the main goal is to deal with inconsistences that can appear among tuples in real-life databases. Experiments on small real-life database showed the feasibility and the interest of our proposal. Encouraging results have been presented, considering synthetic medium-size databases.

This work in integrated in a tool called DBA Companion (Lopes et al., 2004) devoted to logical database tuning (De Marchi et al., 2003). Its objective is to be able to connect any database (independently of the underlying RDBMS) in order to give some insights to DBA/analyst such as the FDs satisfied in a given relation, the inclusion dependencies satisfied by the database or small informative examples of a given database.

References

Abiteboul, S., R. Hull, and V. Vianu: 1995, *Foundations of Databases*. Reading, Mass.: Addison-Wesley.

Afrati, F. N., A. Gionis, and H. Mannila: 2004, 'Approximating a collection of frequent sets'. In: W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel (eds.): *International Conference on Knowledge Discovery and Data Mining (KDD'04), Washington, USA*. pp. 2–19, ACM.

Agrawal, R. and R. Srikant: 1994, 'Fast Algorithms for Mining Association Rules in Large Databases'. In: J. B. Bocca, M. Jarke, and C. Zaniolo (eds.): *International Conference on Very Large Data Bases (VLDB'94), Santiago de Chile, Chile*. pp. 487–499, Morgan Kaufmann.

Albrecht, M., E. Buchholz, A. Düsterhöft, and B. Thalheim: 1995, 'An Informal and Efficient Approach for Obtaining Semantic Constraints Using Sample Data and Natural Language Processing'. In: L. Libkin and B. Thalheim (eds.): *Semantics in Databases*, Vol. 1358 of *Lecture Notes in Computer Science*. pp. 1–28, Springer.

Bauckmann, J., U. Leser, F. Naumann, and V. Tietz: 2007, 'Efficiently Detecting Inclusion Dependencies'. In: *International Conference on Data Engineering (ICDE'07)*. pp. 1448–1450, IEEE Computer Society.

Bay, S. D.: 1999, 'The UCI KDD Archive [http://kdd.ics.uci.edu]'. Technical report, Irvine, CA: University of California, Department of Information and Computer Science.

Bell, S. and P. Brockhausen: 1995, 'Discovery of Constraints and Data Dependencies in Databases (Extended Abstract)'. In: N. Lavrac and S. Wrobel (eds.): *European Conference on Machine Learning (ECML'95), Crete, Greece*, Vol. 912 of *Lecture Notes in Computer Science*. pp. 267–270, Springer.

Calders, T. and J. Wijsen: 2001, 'On Monotone Data Mining Languages'. In: G. Ghelli and G. Grahne (eds.): *International Workshop on Database Programming Languages (DBPL'01), Frascati, Italy*. Springer.

Casanova, M., R. Fagin, and C. Papadimitriou: 1984, 'Inclusion dependencies and their interaction with functional dependencies'. *Journal of Computer and System Sciences* **24**(1), 29–59.

Casanova, M. A., L. Tucherman, and A. L. Furtado: 1988, 'Enforcing Inclusion Dependencies and Referencial Integrity'. In: F. Bancilhon and D. J. DeWitt (eds.): *International Conference on Very Large Data Bases (VLDB'88), Los Angeles, California, USA*. pp. 38–49, Morgan Kaufmann.

Cheng, Q., J. Gryz, F. Koo, T. Y. C. Leung, L. Liu, X. Qian, and B. Schiefer: 1999, 'Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database'. In: M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie (eds.): *International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK*. pp. 687–698, Morgan Kaufmann.

Dasu, T., T. Johnson, S. Muthukrishnan, and V. Shkapenyuk: 2002, 'Mining Database Structure; Or, How to Build a Data Quality Browser'. In: *ACM SIGMOD Conference 2002*. Madison, Wisconsin, USA, pp. 240–251.

De Marchi, F., S. Lopes, and J.-M. Petit: 2002, 'Efficient Algorithms for Mining Inclusion Dependencies'. In: C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, and M. Jarke (eds.): *International Conference on Extending Database Technology (EDBT'02), Prague, Czech Republic*, Vol. 2287 of *Lecture Notes in Computer Science*. pp. 464–476, Springer.

De Marchi, F., S. Lopes, J.-M. Petit, and F. Toumani: 2003, 'Analysis of existing databases at the logical level: the DBA companion project'. *ACM Sigmod Record* **32**(1), 47–52.

De Marchi, F. and J.-M. Petit: 2003, 'Zigzag : a new algorithm for discovering large inclusion dependencies in relational databases'. In: *International Conference on Data Mining (ICDM'03), Melbourne, Florida, USA*. pp. 27–34, IEEE Computer Society.

De Marchi, F. and J.-M. Petit: 2005, 'Approximating a Set of Approximate Inclusion Dependencies'. In: *International Conference on Intelligent Information System (IIS'05)*. Gdansk, Poland, pp. 633–640, Springer-Verlag.

Ganter, B. and R. Wille: 1999, *Formal Concept Analysis*. Springer-verlag.

Gryz, J.: 1998, 'Query Folding with Inclusion Dependencies'. In: *International Conference on Data Engineering (ICDE'98), Orlando, Florida, USA*. pp. 126–133, IEEE Computer Society.

Han, J. and M. Kamber: 2000, *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Huhtala, Y., J. Karkkainen, P. Porkka, and H. Toivonen: 1999, 'TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies'. *The Computer Journal* **42**(2), 100–111.

Kantola, M., H. Mannila, K. J. Raïha, and H. Siirtola: 1992, 'Discovering functional and inclusion dependencies in relational databases'. *International Journal of Intelligent Systems* **7**, 591–607.

Kivinen, J. and H. Mannila: 1995, 'Approximate inference of functional dependencies from relations'. *Theoretical Computer Science* **149**(1), 129–149.

Koeller, A. and E. A. Rundensteiner: 2003, 'Discovery of high-dimentional Inclusion Dependencies (Poster)'. In: *Poster session of International Conference on Data Engineering (ICDE'03)*. IEEE Computer Society.

Levene, M. and G. Loizou: 1999, *A Guided Tour of Relational Databases and Beyond*. Springer.

Levene, M. and M. W. Vincent: 2000, 'Justification for Inclusion Dependency Normal Form'. *IEEE Transactions on Knowledge and Data Engineering* **12**(2), 281–291.

Lopes, S., F. De Marchi, and J.-M. Petit: 2004, 'DBA Companion: A Tool for Logical Database Tuning'. In: *Demo session of International Conference on Data Engineering (ICDE'04)*. `http://www.isima.fr/~demarchi/dbacomp/`, IEEE Computer Society.

Lopes, S., J.-M. Petit, and L. Lakhal: 2002a, 'Functional and Approximate Dependencies Mining: Databases and FCA Point of View'. *Special issue of JETAI* **14**(2/3), 93–114.

Lopes, S., J.-M. Petit, and F. Toumani: 2002b, 'Discovering Interesting Inclusion Dependencies: Application to Logical Database Tuning'. *Information System* **17**(1), 1–19.

Mannila, H. and K.-J. Räihä: 1986, 'Inclusion Dependencies in Database Design'. In: *International Conference on Data Engineering (ICDE'86), Los Angeles, California, USA*. pp. 713–718, IEEE Computer Society.

Mannila, H. and K. J. Raïha: 1994, *The Design of Relational Databases*. Addison-Wesley, second edition.

Mannila, H. and H. Toivonen: 1997, 'Levelwise Search and Borders of Theories in Knowledge Discovery'. *Data Mining and Knowledge Discovery* **1**(1), 241–258.

Miller, R. J., M. A. Hernandez, L. M. Haas, L. Yan, C. T. H. Ho, R. Fagin, and L. Popa: 2001, 'The Clio Project: Managing Heterogeneity'. *ACM SIGMOD Record* **30**(1), 78–83.

Mitchell, J. C.: 1983, 'The Implication Problem for Functional and Inclusion Dependencies'. *Information and Control* **56**(3), 154–173.

Novelli, N. and R. Cicchetti: 2001, 'Functional and Embedded Dependency Inference: a Data Mining Point of View'. *Information System* **26**(7), 477–506.

Sarawagi, S., S. Thomas, and R. Agrawal: 2000, 'Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications'. *Data Mining and Knowledge Discovery* **4**(2/3), 89–125.

Wyss, C., C. Giannella, and E. Robertson: 2001, 'FastFDs: A Heuristic-Driven Depth-First Algorithm for Mining Functional Dependencies from Relation Instances'. In: Y. Kambayashi, W. Winiwarter, and M. Arikawa (eds.): *Data Warehousing and Knowledge Discovery (DaWaK'01), Munich, Germany*, Vol. 2114 of *Lecture Notes in Computer Science*. pp. 101–110.