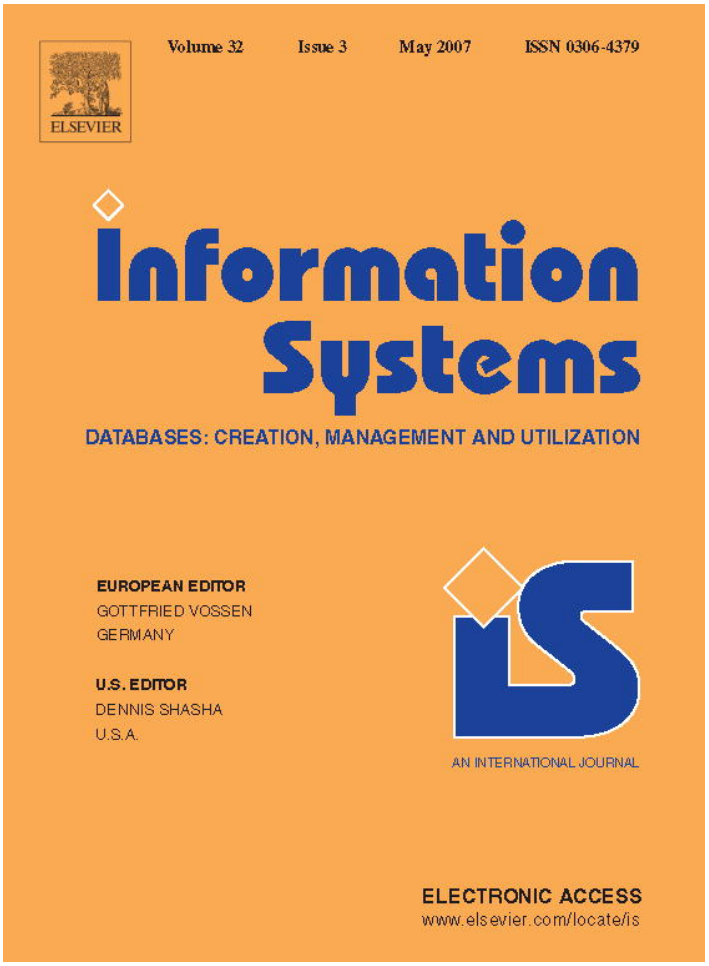

Available online at www.sciencedirect.com
ScienceDirect
Information Systems 32 (2007) 446–457
www.elsevier.com/locate/infosys

Semantic sampling of existing databases through informative Armstrong databases

Fabien De Marchi[a,*], Jean-Marc Petit[b]

[a]Université Lyon 1, bâtiment Nautibus, 8 boulevard Niels Bohr, 69 622 Villeurbanne Cedex, France
[b]INSA Lyon, Bâtiment Blaise PASCAL, 20 Avenue Albert Einstein, 69621 Villeurbanne, France

Received 17 November 2004; received in revised form 21 October 2005; accepted 7 December 2005

Recommended by J. Van den Bussche

Abstract

Functional dependencies (FDs) and inclusion dependencies (INDs) convey most of data semantics in relational databases and are very useful in practice since they generalize keys and foreign keys. Nevertheless, FDs and INDs are often not available, obsolete or lost in real-life databases. Several algorithms have been proposed for mining these dependencies, but the output is always in the same format: a simple list of dependencies, hard to understand for the user. In this paper, we define *informative Armstrong databases* (IADBs) from databases as being small subsets of an existing database, satisfying *exactly* the same FDs and INDs. They are an extension of the classical notion of Armstrong databases, but more suitable for the understanding of dependencies, since tuples are real-world tuples. The main result of this paper is to bound the size of an IADB in the case of non-circular INDs. A constructive proof of this result is given, from which an algorithm has been devised. An implementation and experiments against a real-life database were performed; the obtained database contains 0.6% of the initial database tuples only. More importantly, such semantic sampling of databases appear to be a key feature for the understanding of existing databases at the logical level.

Keywords: Functional dependencies; Inclusion dependencies; Armstrong databases; Database maintenance; Dependency visualization

1. Introduction

Functional dependencies (FDs) and inclusion dependencies (INDs) convey most of data semantics in relational databases [1] and are very popular in practice since they generalize keys and foreign keys. For example, they are necessary to define normal forms [2,3] and are widely used in many database applications such as database design and maintenance [4,3,5], data integration [6], or semantic query optimization [7,8]. A friendly alternative representations of sets of functional and inclusion dependencies are known to be Armstrong databases [9,10]. They are databases verifying exactly a given set of FDs and INDs and their implications. Up to now, they are proved to be mainly useful in database design processes, integrated in "design-by-example" tools [11,12].

In practice, FDs and INDs satisfied by existing databases cannot be defined as such; they have to appear as key or foreign key constraints in the best cases. Moreover, they could have been lost, or never

*Corresponding author.
E-mail addresses: fabien.demarchi@liris.cnrs.fr (F. De Marchi), jean-marc.petit@liris.cnrs.fr (J.-M. Petit).

0306-4379/$ - see front matter © 2006 Elsevier Ltd All rights reserved.
doi:10.1016/j.is.2005.12.007

been defined in case of bad conception. For these reasons, the discovery of satisfied FDs and INDs into an existing database has been considered as a challenging KDD task [13–16]. But, to our knowledge, the result offered to the final user is always the same: a list of dependencies, hard to understand even for a domain expert, typically a database administrator (DBA).

In the framework of this paper, we assume the discovery of FDs and INDs has been already done in order to focus on the final part of the discovery task: the visualization by a DBA of FDs and INDs satisfied by the database. For this purpose, the definition of Armstrong databases seems to be not really adapted, since tuples in such example relations are completely fictitious. Even when an existing database is available, example tuples are constructed (in the best case) using active domain values of attributes, but always randomly chosen [10,17,12,15]. As a consequence, tuples in Armstrong databases do not reflect at all those of the initial database and are not informative enough for a DBA. A convincing example of this kind of problem will be exhibited in the sequel of the paper (see Example 2).

1.1. Paper contribution

We propose in this paper to extend the definition of Armstrong databases, to take into account *FDs and INDs satisfied by a database*. Given an existing database, we define *informative Armstrong databases* (IADBs), which satisfy exactly the same functional and inclusion dependencies as the input database, and in which all tuples come from the input database. The main result of this paper is to bound the size of an IADB in the case of non-circular INDs. A constructive proof of this result is given, from which an algorithm has been devised. Note also that circular INDs are taken into account in our implementation, but the bounds do not apply. An implementation and experiments against a real-life database were performed: the obtained database contains only 0.6% of the initial database tuples.

1.2. Related works

To our knowledge, very few works consider the visualization of FDs and INDs from databases. In [12,15], the classical definition of Armstrong relations (i.e. Armstrong databases considering only FDs) is directly applied to propose example relations to domain expert. The basic idea is to give some help to understand the FDs satisfied by a given relation. The drawback of such a solution in our context comes from the way tuples are built, almost randomly (cf. Section 3). In [21], from an existing database, potential constraints are derived from heuristics considerations; for such a constraint, an example from the database is proposed to the user. The constraint is validated only if the user considers that this example is a valid one. Even if the aim is close to ours, our approach is quite different. Moreover, in our proposal, the user obtains a small database sample, with same FDs and INDs.

1.3. Application

Given an existing database, such database examples could be very useful to better understand data semantics usually conveyed by FDs and INDs. We expect the greatest impact of our work to be for the problem of logical database tuning, e.g. a DBA would have the opportunity to quickly visualize FDs and INDs and then detect data integrity problems [22].

In order to observe IADB users, and getting their feedbacks, we proposed to master students the following case study: "Reverse engineering the database ***movies*** using DBA Companion [23]". Their first attempt was to discover keys and foreign keys using data mining techniques. As soon as their initial intuition was contracted by the discovered constraints, they generated our IADB for ***movies*** to go one step beyond. We observed that, thanks to the small size of the IADB, they were able to perform instantaneous sorts, selections, projections and joins to manipulate and visualize the examples and counter-examples. IADBs were a key feature of their work, and conducted them to understand the logical database design, which is a mandatory step before reverse engineering the database schema.

This work takes place in a project devoted to the logical tuning of existing databases [22,23]: its goal is to be able to connect any database (independently of the underlying DBMS) in order to give some insights to DBA/analyst such as:

- FDs and INDs satisfied by the database,
- small examples of the database, thanks to IADBs.

1.4. Paper organization

Preliminaries are given in Section 2. Armstrong databases for databases are defined in Section 3. IADBs for databases are presented in Section 4;

then, the main theorem which bounds the size of IADBs is given. Experimental results are reported in Section 5, and we conclude in Section 6.

2. Preliminaries

We recall some basic concepts of relational databases theory, helpful for the understanding of the paper, for details see e.g. [24,25]. We follow the notation proposed in [25].

Let A be an *attribute*, the set of all its possible values is called the *domain of* A and is denoted by $Dom(A)$. A relation schema R is a finite set of attributes. A database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a finite set of relation schemas.

A *tuple* over a relation schema R is a mapping $t : R \to \bigcup_{A \in R} Dom(A)$, where $t(A) \in Dom(A)$, $\forall A \in R$. A *relation* r over R is a finite set of tuples over R. A *database* $\mathbf{d} = \{r_1, \ldots, r_n\}$ over a database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a finite set of relations over elements of $\mathbf{R}$.

Let t be a tuple over R, and $X \subseteq R$ a set of attributes. The *projection* of t onto X, denoted by $t[X]$, is the restriction of t over X. The projection of a relation r over R onto X, denoted as $\pi_X(r)$, is defined by $\pi_X(r) = \{t[X] \mid t \in r\}$. The *active domain* of an attribute $A \in R$ is the set $\pi_A(r)$.

Let t and u be tuples of a relation r over a schema R, and $X \subseteq R$ an attribute set. We denote by $ag(t,u)$ the set: $ag(t,u) = \{A \in R \mid t[A] = u[A]\}$. The *agree sets of* r, denoted by $ag(r)$ are defined by $ag(r) = \{ag(t,u) \mid t,u \in r, t \neq u\}$.

2.1. Functional dependencies

A *functional dependency* (FD) over a relational schema R is an expression $X \to Y$ where $X, Y \subseteq R$. $X \to Y$ *is satisfied* in a relation r over R (denoted by $r \models X \to Y$) if and only if $\forall t,u \in r, t[X] = u[X] \Rightarrow t[Y] = u[Y]$. We call F_r the set of *all* FDs satisfied by a relation r.

Let F be a set of functional dependencies over R. An FD $X \to Y$ is said to be logically implied by F, denoted $F \models X \to Y$, if $\forall r$ over R, if $r \models F$ then $r \models X \to Y$. The closure of a attribute set X with respect to F is the set: $X_F^+ = \{A \in R \mid F \models X \to A\}$. A set $X \subseteq R$ is *closed* if and only if $X_F^+ = X$. We note $CL(F)$ the family of closed sets induced by F and $GEN(F)$ the unique minimal sub-family of generators in $CL(F)$ such that each member of $CL(F)$ can be expressed as an intersection of sets in $GEN(F)$ [26].

2.2. Inclusion dependencies

An *inclusion dependency* (IND) over a database schema $\mathbf{R}$ is an expression of the form $R[X] \subseteq S[Y]$, where $R, S \in \mathbf{R}$, X and Y are sequences of attributes, respectively, in R and S. Moreover, X and Y are sequences of same size. $R[X] \subseteq S[Y]$ *is satisfied* in a database $\mathbf{d}$ over $\mathbf{R}$ (denoted by $\mathbf{d} \models R[X] \subseteq S[Y]$) if and only if $\forall t \in r, \exists u \in s$ such that $t[X] = u[Y]$ (or equivalently $\pi_X(r) \subseteq \pi_Y(s)$).

A set of INDs I is said to be *circular* if (1) it contains an IND $R[X] \subseteq R[Y]$ or (2) it contains $m (m > 1)$ INDs of the form: $R_1[X_1] \subseteq R_2[Y_2]$, $R_2[X_2] \subseteq R_3[Y_3], \ldots, R_m[X_m] \subseteq R_1[Y_1]$. If $X_1 = Y_1$ I is said to be *proper circular* [25].

Let I be a set of INDs over $\mathbf{R}$. An IND $R[X] \subseteq S[Y]$ is said to be logically implied by I, denoted $I \models R[X] \subseteq S[Y]$, whenever for each $\mathbf{d}$ over $\mathbf{R}$, if $\mathbf{d} \models I$ then $\mathbf{d} \models R[X] \subseteq S[Y]$.

Let I and J be two sets of INDs. I is *a cover* of J if and only if $I^+ = J^+$, with $I^+ = \{R[X] \subseteq S[Y]$ s.t. $I \models R[X] \subseteq S[Y]\}$.

Given $i = R[X] \subseteq S[Y]$ and $j = R[X'] \subseteq S[Y']$ two INDs over a database schema $\mathbf{R}$, we say that i *generalizes* j (or j *specializes* i), denoted by $i \preceq j$, if $X' = \langle A_1, \ldots, A_n \rangle$, $Y' = \langle B_1, \ldots, B_n \rangle$, and there exists a set of indices $k_1 < \cdots < k_l \in \{1, \ldots, n\}$ with $l \leqslant n$ such that $X = \langle A_{k_1}, \ldots, A_{k_l} \rangle$ and $Y = \langle B_{k_1}, \ldots, B_{k_l} \rangle$ [27,28].

Example 1. For example, $R[AC] \subseteq S[DF] \preceq R[ABC] \subseteq S[DEF]$, but $R[AB] \subseteq S[DF] \not\preceq R[ABC] \subseteq S[DEF]$.

Let $\mathbf{d}$ be a database over $\mathbf{R}$. The IND satisfaction is anti-monotone, i.e. if $\mathbf{d} \not\models i$ and $i \preceq j$, then $\mathbf{d} \not\models j$. Thus, each set I of INDs can be expressed by its *positive border* denoted by $\mathscr{B}d^+(I)$, which is the set of the most specialized elements in I, or by its *negative border* $\mathscr{B}d^-(I)$, the set of the most generalized elements not in I [27].

2.3. Armstrong databases for FDs and INDs

Armstrong relations were first defined for FDs [9], and extended to *Armstrong databases* for FDs and INDs [10].

Given a database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$, F a set of FDs and I a set of INDs defined over $\mathbf{R}$, an *Armstrong database* $\mathbf{d} = \{r_1, \ldots, r_n\}$ over $\mathbf{R}$ for $F \cup I$ is such that for all FDs or INDs α, we have: $\mathbf{d} \models \alpha \Longleftrightarrow F \cup I \models \alpha$.

In the case of FDs alone, a characterization of Armstrong relations is given in [17]: if r and F are, respectively, a relation and a set of FDs over a schema R, then r is an Armstrong relation for F if and only if $GEN(F) \subseteq ag(r) \subseteq CL(F)$.

An Armstrong database exists for all sets of (standard) FDs and INDs [10]. Algorithms were proposed for computing Armstrong databases in the case of not circular or proper circular set of INDs [12,29].

2.4. Notations

Given a relation r, the number of tuples in r, denoted by $|r|$, is called the *size of* r. Given a database $\mathbf{d} = \{r_1, \ldots, r_n\}$, the number of tuples in $\mathbf{d}$, denoted by $|\mathbf{d}|$, is called the *size of* $\mathbf{d}$ and is equal to $\sum_{i=1}^{n} |r_i|$.

Finally, for convenience, we note $F_{\mathbf{d}}$ and $I_{\mathbf{d}}$, respectively, the set of *all* FDs and INDs satisfied by $\mathbf{d}$ over $\mathbf{R}$, i.e. $I_{\mathbf{d}} = \{R[X] \subseteq S[Y] \mid \mathbf{d} \models R[X] \subseteq S[Y]\}$ and $F_{\mathbf{d}} = \cup_{r \in \mathbf{d}} F_r$ (up to a renaming of attribute names in relation schema).

3. Armstrong databases for databases

Armstrong databases for databases are a natural extension of Armstrong databases for FDs and INDs: the FDs and INDs considered are simply the ones satisfied by the original database.

Definition 1. Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database schema and $\mathbf{d}$ a database over $\mathbf{R}$. An Armstrong database for $\mathbf{d}$ is a database $\overline{\mathbf{d}}$ over $\mathbf{R}$ such that $\overline{\mathbf{d}}$ is an Armstrong database for $F_{\mathbf{d}} \cup I_{\mathbf{d}}$.

Thus, an Armstrong database for a database $\mathbf{d}$ is an alternative representation of the dependencies satisfied by $\mathbf{d}$. This definition raises several questions:

(1) Database is an Armstrong database for itself, what about the size of an Armstrong database with respect to the size of the initial database?
(2) What is the "semantic power" of such databases?

3.1. Limits of existing solutions

Armstrong databases were showed helpful for design by examples of databases [11,12]. Giving a schema and a user-defined set of constraints, an Armstrong database can be generated (also called *example database*) so that the designer can visualize the dependencies through this example. She/he can decide to modify the example database, and then dependencies are recalculated until the designer agrees with the example database and the set of constraints.

Our framework is different since we are interested in database understanding, occurring quite often in practice; either for database maintenance or during a KDD process. In this setting, a naive approach would be: (1) discovering FDs and INDs satisfied by the input database and (2) constructing Armstrong databases from these dependencies.[1] In this case, the computed database should be very small in practice, according to the bounds exhibited in [17], but suffers an important drawback pointed out by the following example.

Example 2. The database ***movies*** (Fig. 1) will be used as a running example throughout the paper. This toy sample comes from a real-life database [30] (presented in details in Section 5). The relation *movies* stores a set of movies; if a movie is a remake of other ones, it is reported in relation remakes. The column *FRAC* in remakes aims at storing the similarity rate between a movie and its remake.

We first consider the relation remakes only. To understand remakes, the set of satisfied FDs has to be discovered. We get: $F = \{A \longrightarrow B, B \longrightarrow AC, CE \longrightarrow B, DE \longrightarrow C, AE \longrightarrow D, AD \longrightarrow E\}$[2] and we want to produce an Armstrong relation for F. Several methods exist [10,17,31], they all construct tuples either from the integer set or from the (active) attribute domains. Fig. 2 gives such an Armstrong relation, which could be obtained from existing algorithms using the active domains of attributes [15].

An analyst (DBA or data miner person) may visualize that A is not a key, from the two first tuples. Nevertheless, these two tuples do not carry semantic information, and do not give clues to the DBA to understand why A is not a key. We shall see in the sequel (cf. Example 3) how to tackle this problem.

To cope with this limitation, we propose to define and compute *informative Armstrong databases* for a given database.

[1] Obviously, if the set of INDs is not proper circular, such database cannot be computed using state of the art algorithms.

[2] F is a cover of FDs that hold in *remakes*, i.e. an equivalent representation of FDs satisfied by *remakes*.

remakes

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	The journey	1959	0.9	11	Mrs Fifi
29	Cléopâtre	1962	0.3	23	Cléopâtre
28	Courageous sisters	1938	0.9	17	Four sisters
15	Carmen loves	1948	0.2	22	Gilda
21	One hour with you	1932	0.9	18	The wedding ring
16	The journey	1959	0.3	17	Fours sisters
40	L'Oeuvre au noir	1990	0.8	42	The name of the rose
12	Young at heart	1938	0.9	30	Casablanca
40	L'Oeuvre au noir	1990	0.1	14	Thérèse

movies

FILMID (G)	*TITLE* (H)	*YEAR* (I)
11	Mrs Fifi	1944
12	Young at heart	1938
13	Exorcist	1971
43	Thérèse	1980
15	Carmen loves	1948
16	The journey	1959
17	Four sisters	1938
18	The wedding ring	1924
21	One hour with you	1932
22	Gilda	1946
23	Cléopâtre	1934
28	Courageous sisters	1938
29	Cléopâtre	1962
30	Casablanca	1944
40	L'Oeuvre au noir	1990
41	The manitou	1978
42	The name of the rose	1985

Fig. 1. The ***movies*** database.

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	The journey	1959	0.9	11	Mrs Fifi
16	The journey	1959	0.3	23	Cléopâtre
29	Cléopâtre	1959	0.9	17	Four sisters
28	Courageous sisters	1962	0.9	22	Gilda
15	Carmen loves	1938	0.2	11	The wedding ring

Fig. 2. An Armstrong relation for *remakes*.

4. Informative Armstrong databases

An IADB is an Armstrong database, but with an additional constraint: all the tuples have to be real tuples from the input database, rather than being artificially constructed.

In the following, we extend the classical definition of operator $\subseteq$ to databases: Let $\mathbf{d} = \{r_1, \ldots, r_n\}$ and $\overline{\mathbf{d}} = \{\overline{r_1}, \ldots, \overline{r_n}\}$ be two databases over the same database schema. We assume without loss of generality that relation schemas are arranged in some order (indices here). We say that $\overline{\mathbf{d}}$ is included in $\mathbf{d}$, denoted by $\overline{\mathbf{d}} \subseteq \mathbf{d}$, if for all $i = 1..n$ $\overline{r}_i \in \overline{\mathbf{d}}$, $\overline{r}_i \subseteq r_i$.

Definition 2 (*Informative Armstrong databases*). Given a database $\mathbf{d} = \{r_1, \ldots, r_n\}$ over a schema

$\mathbf{R} = \{R_1, \ldots, R_n\}$, an IADB $\overline{\mathbf{d}}$ for $\mathbf{d}$ is defined over $\boldsymbol{R}$ and is such that:

- $\overline{\mathbf{d}}$ is an Armstrong database for $\mathbf{d}$ and
- $\overline{\mathbf{d}} \subseteq \mathbf{d}$.

Thus, the idea is to highlight a subset of each relation in $\mathbf{d}$, keeping exactly the same FDs in each relation and the same INDs between relations. Thus, tuples in IADBs will reflect most of the semantic of the initial data, keeping a coherent meaning of the underlying "real world".

4.1. Toward "small" informative Armstrong databases

The existence of IADBs is ensured by the fact that any database is an IADB for itself. Of course the interest would be to construct IADBs whose size is less than the size of the input database. The main result of this paper is given in Theorem 1. Roughly speaking, the size of any IADB for a database $\mathbf{d}$ can be bounded using: (1) the generator sets of FDs satisfied by each relation in $\mathbf{d}$, (2) the positive border of INDs satisfied by $\mathbf{d}$, and (3) the negative border of INDs not satisfied by $\mathbf{d}$.

We denote by $\mathscr{B}d^-(I_d)_{R\to S}$ the negative border of INDs from R to S satisfied by $\mathbf{d}$, i.e. $\mathscr{B}d^-(I_d)_{R\to S} = \{R[X] \subseteq S[Y] \in \mathscr{B}d^-(I_d)\}$. A similar notation $\mathscr{B}d^+(I_d)_{R\to S}$ is used for the positive border.

Theorem 1. *Let* $\mathbf{d} = \{r_1, \ldots, r_n\}$ *be a database over a schema* $\mathbf{R} = \{R_1, \ldots, R_n\}$. *If* $\mathscr{B}d^+(I_d)$ *is non-circular, an IADB* $\overline{\mathbf{d}} = \{\overline{r_1}, \ldots, \overline{r_n}\}$ *for* $\mathbf{d}$ *exists, such that* $|\overline{\mathbf{d}}| = \sum_{i=1}^{n} |\overline{r}_i|$ *with,* $\forall i = 1, \ldots, n$:

$$\left\lceil \frac{1 + \sqrt{1 + 8 \times |GEN(F_{r_i})|}}{2} \right\rceil \leqslant |\overline{r}_i|$$

and

$$|\overline{r}_i| \leqslant 2 \times |GEN(F_{r_i})| + \sum_{j=1}^{n} |\mathscr{B}d^-(I_d)_{R_i \to R_j}| + \sum_{j=1}^{n} |\mathscr{B}d^+(I_d)_{R_j \to R_i}| \times |\overline{r}_j|.$$

Note that the upper bound is recursively defined, and then relations have to be considered in a specific order. Such a partial order exists and is easily computable if $\mathscr{B}d^+(I_d)$ is non-circular.

To prove this theorem, we exhibit a constructive proof whose main parts are explained through Algorithm 1.

Algorithm 1. Informative Armstrong database
Input: $\mathbf{d}$, $GEN(F_d)$, $\mathscr{B}d^+(I_d)$ and $\mathscr{B}d^-(I_d)$;
Output: An IADB $\overline{\mathbf{d}}$ for $\mathbf{d}$;
1: $\overline{\mathbf{d}} = \emptyset$;
2: **for all** $r \in \mathbf{d}$ **do**
3: Compute $GEN(F_r)$;
4: $\overline{\mathbf{d}} = \overline{\mathbf{d}} \cup GenIAR(r, GEN(F_r))$—*Dealing with satisfied and not satisfied FDs*
5: **end for**
6: $\overline{\mathbf{d}} = InvalidIND(\overline{\mathbf{d}}, \mathscr{B}d^-(I_d), \mathbf{d})$;—*Dealing with not satisfied INDs*
7: $\overline{\mathbf{d}} = ValidIND(\overline{\mathbf{d}}, \mathscr{B}d^+(I_d), \mathbf{d})$;—*Dealing with satisfied INDs*
8: **Return** $\overline{\mathbf{d}}$.

The functions *GenIAR*, *InvalidIND* and *ValidIND* are defined latter on.

This algorithm takes as input a database $\mathbf{d}$, the generator sets of FDs satisfied by $\mathbf{d}$, and the positive and negative borders of INDs satisfied by $\mathbf{d}$. An IADB for $\mathbf{d}$ is then computed through three steps; each one consists in adding tuples from $\mathbf{d}$ into the database under construction, in order to obtain a database $\overline{\mathbf{d}}$ such that $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ and $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$. As shown by the next property, this is a sufficient condition for $\overline{\mathbf{d}}$ to be an IADB for $\mathbf{d}$.

Property 1. *Let* $\mathbf{d}$ *and* $\overline{\mathbf{d}}$ *be two databases over a schema* $\mathbf{R}$ *such that* $\overline{\mathbf{d}} \subseteq \mathbf{d}$. $\overline{\mathbf{d}}$ *is an IADB for* $\mathbf{d}$ *if and only if* $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ *and* $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$.

Proof. It is sufficient to show that $F_{\mathbf{d}}$ and $I_{\mathbf{d}}$ do not have interaction, i.e. no dependence can be implied by $F_d \cup I_d$ if it is not implied by F_d or by I_d.

Let α be a dependence (IND or FD) such that $F_{\mathbf{d}} \cup I_{\mathbf{d}} \vDash \alpha$. Since $d \vDash F_{\mathbf{d}}$ and $d \vDash I_{\mathbf{d}}$, we have $d \vDash \alpha$. Since $F_{\mathbf{d}}$ and $I_{\mathbf{d}}$ are the set of *all* FDs and INDs satisfied by d, we have $\alpha \in F_{\mathbf{d}}$ or $\alpha \in I_{\mathbf{d}}$. Thus $F_{\mathbf{d}}$ and $I_{\mathbf{d}}$ have no interaction. □

In the sequel, each step of Algorithm 1 is detailed and illustrated with the running example. Finally, we give the proof of Theorem 1.

4.2. Dealing with FDs only

Given a database $\mathbf{d}$, the function *GenIAR* aims at computing a database $\overline{\mathbf{d}}$ such that (1) $\overline{\mathbf{d}} \subseteq \mathbf{d}$ and (2)

$F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$. We propose to achieve this task by computing, for each relation r_i in $\mathbf{d}$, a so-called *informative Armstrong relation* (IAR) $\overline{r}_i$ such that: (1) $\overline{r}_i \subseteq r_i$ and (2) $F_{\overline{r}_i} = F_{r_i}$.

The following lemma characterizes exactly IARs, and gives some insights on their construction. A similar result was given in [19].

Lemma 1. *Let R be a relation schema, r a relation over R, and $GEN(F_r)$ the family of generators of the closed sets w.r.t. F_r. A subset $\overline{r}$ of r is an IAR for r if and only if $GEN(F_r) \subseteq ag(\overline{r})$.*

Proof. ($\Leftarrow$) Since $GEN(F_r) \subseteq ag(\overline{r})$, it remains to show that $ag(\overline{r}) \subseteq CL(F_r)$ (cf. preliminaries in Section 2). $\overline{r} \subseteq r$ thus $ag(\overline{r}) \subseteq ag(r)$. Since r is an Armstrong relation for F_r, $ag(r) \subseteq CL(F_r)$ and then $ag(\overline{r}) \subseteq CL(F_r)$.

Moreover, we have $\overline{r} \subseteq r$, thus $\overline{r}$ is an IAR for r.

($\Rightarrow$) $\overline{r}$ is an Armstrong relation for F_r, and $GEN(F_r) \subseteq ag(\overline{r}) \subseteq CL(F_r)$. □

An algorithm can be deduced from this characterization. It takes randomly two tuples in r that agree on each generator of F_r, and inserts them into the IAR under construction. Such an algorithm is linear in the number of generator sets of F_r and is referred as $GenIAR(r, Gen(F_r))$ in line 4 of Algorithm 1.

Example 3. Fig. 3 gives the database $\overline{\textbf{\textit{movies}}}$, made up of informative Armstrong relations for each relation of ***movies*** (Fig. 1). These relations are computed from the following generator sets: $\{ABC, CD, D, E\}$ for *remakes* and $\{H, I\}$ for movies. Note that drawbacks mentioned in the Example 2 do not longer exist, since the two remakes whose *ID* is equal to 16 correspond to "real-life" remakes, and thus make sense for the analyst. At the same time, the analyst can observe that attribute *PRID* does not determine attribute *PRTITLE*. The third tuple ("Fours sisters" in place of "Four sisters") suggest syntax errors.

From Lemma 1, we can deduce bounds for the size of each relation $\overline{r}_i$, from which a bound for the size of the database $\overline{\mathbf{d}}$ can be easily derived.

Lemma 2. *Let $\mathbf{d} = \{r_1, \ldots, r_n\}$ be a database. There exists a database $\overline{\mathbf{d}} \subseteq \mathbf{d}$ such that $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ and $|\overline{\mathbf{d}}| = \sum_{i=1}^{n} |\overline{r}_i|$ with, $\forall i = 1, \ldots, n$:*

$$\left\lceil \frac{1 + \sqrt{1 + 8 \times |GEN(F_{r_i})|}}{2} \right\rceil \leqslant |\overline{r}_i| \leqslant 2 \times |GEN(F_{r_i})|.$$

Proof. The lower bound is due to [17]. Intuitively, it corresponds to the minimum number of tuples in order to have one pair of tuples that agrees on each generator set of F_{r_i}. The upper bound is reached if all these pairs are distinct. □

The next lemma ensures that any tuple of $\mathbf{d}$ can be inserted into $\overline{\mathbf{d}}$ without changing the set of FDs in $\overline{\mathbf{d}}$, and thus ensures the correctness of our proposition w.r.t. FDs.

Lemma 3. *Let $\mathbf{d}$ be a database and $\overline{\mathbf{d}} \subseteq \mathbf{d}$ such that $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$. For every database $\mathbf{b}$ such that $\overline{\mathbf{d}} \subseteq \mathbf{b} \subseteq \mathbf{d}$, we have $F_{\mathbf{b}} = F_{\mathbf{d}}$.*

Proof. Let $r_i \in \mathbf{d}$, $\overline{r}_i \in \overline{\mathbf{d}}$ and $b_i \in \mathbf{b}$ such that $\overline{r}_i \subseteq b_i \subseteq r_i$. From Lemma 1, $GEN(F_{r_i}) \subseteq ag(\overline{r}_i)$.

Suppose $\overline{r}_i \subseteq b_i \subseteq r_i$. We have $ag(\overline{r}_i) \subseteq ag(b_i)$, and thus $GEN(F_{r_i}) \subseteq ag(b_i)$.

$\overline{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	The journey	1959	0.9	11	Mrs Fifi
28	Courageous sisters	1938	0.9	17	Four sisters
16	The journey	1959	0.3	17	Fours sisters
12	Young at heart	1938	0.9	30	Casablanca

$\overline{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	Courageous sisters	1938
29	Cléopâtre	1962

Fig. 3. The database $\overline{\textbf{\textit{movies}}}$ for FDs only.

Since $b_i \subseteq r_i$, b_i is an IAR for r_i according to Lemma 1. □

4.3. Dealing with non-satisfied INDs

The aim of the second step (function *InvalidIND* in Algorithm 1) is to add tuples from $\mathbf{d}$ into $\overline{\mathbf{d}}$, in order to violate INDs not satisfied by $\mathbf{d}$. Recall that $I_{\mathbf{d}}$ represents INDs satisfied by $\mathbf{d}$. Thus, we compute $\overline{\mathbf{d}}$ such that

- $\overline{\mathbf{d}} \subseteq \mathbf{d}$,
- $I_{\overline{\mathbf{d}}} \subseteq I_{\mathbf{d}}$.

To ensure that an IND is violated, we define the simple notion of *disqualifying tuple* as follows:

Definition 3 (*Disqualifying tuple*). Let $\mathbf{d} = \{r_1, \ldots, r_n\}$ be a database over a schema $\mathbf{R} = \{R_1, \ldots, R_n\}$, and $R_i[X] \subseteq R_j[Y]$ an IND over $\mathbf{R}$. A tuple $t \in r_i$ is a *disqualifying tuple* for $R_i[X] \subseteq R_j[Y]$ if $t[X] \notin \pi_Y(r_j)$.

Therefore, we have to insert into $\overline{\mathbf{d}}$ only one disqualifying tuple for each IND i such that $i \notin I_{\mathbf{d}}$. In fact, thanks to the anti-monotony property of IND satisfaction, it is enough to violate the negative border of $I_{\mathbf{d}}$, i.e. $\mathscr{B}d^-(I_d)$. The function $InvalidIND(\mathbf{d}, \mathscr{B}d^-(I_d), \overline{\mathbf{d}})$, line 6 of Algorithm 1, performs this task. For each IND in $\mathscr{B}d^-(I_d)$, if a disqualifying tuple does not belong to the database under construction, such a tuple is elicited in $\mathbf{d}$ and is inserted in $\overline{\mathbf{d}}$. Clearly, such an algorithm is linear in the size of $\mathscr{B}d^-(I_d)$.

Example 4. On the running example, the database $\overline{\boldsymbol{movies}}$ is first initialized to the database $\overline{\boldsymbol{movies}}$ of Fig. 3.

$\mathscr{B}d^-(I_{\boldsymbol{movies}})$ is made up of 12 INDs, not listed here. Let us consider two significant examples of INDs in $\mathscr{B}d^-(I_{\boldsymbol{movies}})$:

- $A \subseteq E$: since $\pi_A(\overline{remakes}) - \pi_E(remakes)$ is not empty, a disqualifying tuple for this IND is already into $\overline{remakes}$.
- $H \subseteq B$: $\pi_H(\overline{\boldsymbol{movies}}) - \pi_B(remakes)$ is empty, a disqualifying tuple has to be chosen in *movies*, for example $\langle 11, MrsFifi, 1944\rangle$, and has to be inserted into $\overline{\boldsymbol{movies}}$.

Finally, we obtain the database $\overline{\boldsymbol{movies}}$ represented in Fig. 4, such that $\overline{\boldsymbol{movies}} \subseteq \boldsymbol{movies}$, $F_{\overline{\boldsymbol{movies}}} = F_{\boldsymbol{movies}}$ and $I_{\overline{\boldsymbol{movies}}} \subseteq I_{\boldsymbol{movies}}$.

Note that such a solution is not unique; in general, many disqualifying tuples can be chosen to enforce that an IND is violated. Nevertheless, a bound for the size of $\overline{\mathbf{d}}$ can be given.

Lemma 4. *Let* $\mathbf{d}$ *be a database. There exists a database* $\overline{\mathbf{d}} \subseteq \mathbf{d}$, *such that* $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ *and* $I_{\overline{\mathbf{d}}} \subseteq I_{\mathbf{d}}$ *and*:

$\overline{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	The journey	1959	0.9	11	Mrs Fifi
28	Courageous sisters	1938	0.9	17	Four sisters
16	The journey	1959	0.3	17	Fours sisters
12	Young at heart	1938	0.9	30	Casablanca

$\overline{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	Courageous sisters	1938
29	Cléopâtre	1962
11	Mrs Fifi	1944

Fig. 4. The database $\overline{\boldsymbol{movies}}$ for FDs and not satisfied INDs.

$|\overline{\mathbf{d}}| = \sum_{i=1}^{n} |\overline{r}_i|$ *with*, $\forall i = 1, \ldots, n$:

$$\left\lceil \frac{1 + \sqrt{1 + 8 \times |GEN(F_{r_i})|}}{2} \right\rceil \leqslant |\overline{r}_i| \leqslant 2 \times |GEN(F_{r_i})| + \sum_{j=1}^{n} |\mathscr{B}d^{-}(I_d)_{R_i \to R_j}|.$$

Proof. The lower bound corresponds to the best case which occurs when disqualifying tuples already belong to $\overline{\mathbf{d}}$ (cf. Lemma 2).

The upper bound is reached if the upper bound for $\overline{\mathbf{d}}$ has already been reached when considering only FDs (Lemma 2), and if one disqualifying tuple has to be inserted for each IND in $\mathscr{B}d^{-}(I_d)$. □

The next lemma ensures that properties of $\overline{\mathbf{d}}$ still hold if new tuples from $\mathbf{d}$ are inserted into $\overline{\mathbf{d}}$.

Lemma 5. *For all database* $\mathbf{b}$ *such that* $\overline{\mathbf{d}} \subseteq \mathbf{b} \subseteq \mathbf{d}$, *we have* $F_{\mathbf{b}} = F_{\mathbf{d}}$ *and* $I_{\mathbf{b}} \subseteq I_{\mathbf{d}}$.

Proof. $F_{\mathbf{b}} = F_{\mathbf{d}}$ is entailed by Lemma 3. Since $\overline{\mathbf{d}} \subseteq \mathbf{b}$, there is in $\mathbf{b}$ at least one disqualifying tuple for each IND in $\mathscr{B}d^{-}(I_d)$, and thus $I_{\mathbf{b}} \subseteq I_{\mathbf{d}}$. □

4.4. Dealing with satisfied INDs

The aim of the last step (function *ValidIND* in Algorithm 1) is to add new tuples from $\mathbf{d}$ in $\overline{\mathbf{d}}$ in order to enforce INDs satisfied by $\mathbf{d}$. Thus, we compute a database $\overline{\mathbf{d}}$ such that $\overline{\mathbf{d}}$ is an IADB for $\mathbf{d}$, i.e. $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ and $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$.

The principle is to consider one by one each IND i in $\mathscr{B}d^{+}(I_d)$ and to check whether or not i is satisfied by the database under construction. If not, i has to be "enforced", by inserting some tuples from $\mathbf{d}$ into $\overline{\mathbf{d}}$.

In the next sections, we consider the case where $\mathscr{B}d^{+}(I_d)$ is circular and the one where it is not.

4.4.1. $\mathscr{B}d^{+}(I_d)$ non-circular

Let us consider that an IND $i = R_i[X] \subseteq R_j[Y]$ has been enforced in $\overline{\mathbf{d}}$. Since we never drop tuples, the only way to violate i is to add tuples into the relation r_i over the left-hand side schema R_i of i. This action occurs only if an IND entering in R_i has to be enforced. Thus, if a relation schema R_i has no incoming IND in $\mathscr{B}d^{+}(I_d)$, every IND from R_i to another schema can be definitively enforced.

The function referred to as *ValidIND*$(\overline{\mathbf{d}}, \mathscr{B}d^{+}(I_d), \mathbf{d})$ line 8 of Algorithm 1 is based on this principle. We consider the partial order induced by INDs between relation schemas, i.e. R_1 precedes R_2 if there is an IND from R_1 to R_2. Each IND whose left-hand side is a relation schema with no predecessor is enforced and can be safely removed from $\mathscr{B}d^{+}(I_d)$. This process is iterated until $\mathscr{B}d^{+}(I_d)$ is empty; this termination condition is ensured by the fact that $\mathscr{B}d^{+}(I_d)$ is non-circular. Such an algorithm has a linear complexity with respect to the size of $\mathscr{B}d^{+}(I_d)$.

Example 5. In our example, the database $\overline{\textbf{\textit{movies}}}$ is first initialized to $\overline{\textbf{\textit{movies}}}$ of Fig. 4. $\mathscr{B}d^{+}(I_{\textbf{\textit{movies}}})$ is made up of 2 INDs: $\mathscr{B}d^{+}(I_{\textbf{\textit{movies}}}) = \{ABC \subseteq GHI, E \subseteq G\}$. The relation *remakes* is first considered, since it does not have incoming IND. The IND $ABC \subseteq GHI$ is enforced by the insertion of the tuple $\langle 16, \text{Le voyage}, 1959\rangle$ into $\overline{\textbf{\textit{movies}}}$, and then $E \subseteq G$ by the insertion of $\langle 17, \text{Four sisters}, 1938\rangle$ and $\langle 30, \text{casablanca}, 1944\rangle$ into $\overline{\textbf{\textit{movies}}}$.

The two INDs are then dropped, and the algorithm terminates. We obtain the database $\overline{\textbf{\textit{movies}}}$ which is an IADB for $\mathbf{d}$, represented in Fig. 5.

We are now able to give a formal proof of Theorem 1.

Proof of Theorem 1. Theorem 1 can be deduced from bounds given in Lemma 4 for $\overline{\mathbf{d}}$ considering FDs and not satisfied INDs only. Recall that, to enforce satisfied INDs we only add tuples from $\mathbf{d}$.

The lower bound can be reached if each IND in $\mathscr{B}d^{+}(I_d)$ is already satisfied by $\overline{\mathbf{d}}$ for FDs and not satisfied INDs only (Lemma 4). The upper bound is reached if, to enforce each IND $R[X] \subseteq S[Y]$ of $\mathscr{B}d^{+}(I_d)$, one tuple has to be inserted into the relation $\overline{s}$ for each tuple into the relation $\overline{r}$. □

4.4.2. $\mathscr{B}d^{+}(I_d)$ circular

The case of non-circular IND is easy to handle since each step is definitive, i.e. each element is considered only once. Unfortunately, things are different in the case of circular INDs which may exist in $\mathscr{B}d^{+}(I_d)$: enforcing an incoming IND into a relation r may violate another IND previously enforced if they belong to the same cycle. In that case, Theorem 1 does not apply. Nevertheless, a simple solution does exist in practice: repeating the enforcement of INDs until a stable state is reached. Even if we are unable to predict the number of iterations, we can guarantee that no infinite loop may arise since any database is by definition an IADB for itself.

To the best of our knowledge, even if classical Armstrong databases exist for all set of FDs and

$\overline{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	The journey	1959	0.9	11	Mrs Fifi
28	Courageous sisters	1938	0.9	17	Four sisters
16	The journey	1959	0.3	17	Fours sisters
12	Young at heart	1938	0.9	30	Casablanca

$\overline{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	Courageous sisters	1938
29	Cléopâtre	1962
11	Mrs Fifi	1944
16	The journey	1959
17	Four sisters	1938
30	Casablanca	1944

Fig. 5. The database $\overline{\textbf{\textit{movies}}}$, IADB for ***movies***.

INDs, no algorithm does exist to compute them in the case of circular set of INDs.

4.4.3. Discussion

A natural question arises concerning the relevance of the upper bound given in Theorem 1: can we expect small sizes for IADBs in practice? Two parameters influence the size of IADBs: the number of generator sets for FDs in each relation, and the size of negative and positive borders for INDs in the database. The first one is known to be potentially exponential in the number of attributes [17]. The second one is also exponential in the number of attributes. Indeed, IND discovery is a problem "representable as sets" [27,16] and thus the borders of INDs are isomorph to the borders of frequent itemsets, which may be exponential in the number of attributes. However, these worst cases correspond to highly artificial databases. For FDs, all existing real-life and synthetic experiments have reported a small number of generator sets [15]. For INDs, the borders are also small in practice (more specially the positive border), since INDs of size greater than 4 or 5 are quite rare in operational databases [32]. Experiments in the next section tend to confirm this discussion.

5. Experiments

The aim of this section is twofold: showing the feasibility of our approach and illustrating the power of reduction of IADBs on a medium-size real-life database.

The database ***movies*** is part of the *UCI KDD Archive* [30]. The original format is plain text, and thus minor changes were necessary to import the data into an Oracle 9i database. The largest relation is made up of 40,000 tuples and the overall cardinality of the database is 68,262. Algorithms were implemented using the C++ language and STL (Standard Template Library), and executed on a Pentium II 500 MHz processor, 392 Mbytes of main memory. For each step of Algorithm 1, we have reported in Fig. 6 the size of the IADB under construction. The last two columns are the lower and upper bounds given in Theorem 1.

As one can see in Fig. 6, we are ensured to compute an IADB with a maximum of 5784 tuples. This is an important result since it corresponds to a reduction of ***movies*** by a factor of 11.8.

Step 1 (FDs): We have computed $\overline{\textbf{\textit{movies}}}$ such that $F_{\overline{\textbf{\textit{movies}}}} = F_{\textbf{\textit{movies}}}$. The size of $\overline{\textbf{\textit{movies}}}$ is equal to 374, which corresponds to a reduction by 182 with exactly the same FDs.

Step 2 (not satisfied INDs): In the database ***movies***, there are 896 INDs in the negative border of satisfied INDs. In our experiments, no disqualifying tuple at all had to be inserted.

Step 3 (satisfied INDs): As seen before, it is enough to enforce, in the database under construction, the positive border of satisfied INDs in ***movies***. Here, the positive border is made up of 10 INDs.

movies	$\overline{\textbf{movies}}$ after step 1	$\overline{\textbf{movies}}$ after step 2	$\overline{\textbf{movies}}$ after step 3	L. bound	U. bound
68262	374	374	434	62	5784

Fig. 6. Experimental results (number of tuples).

Enforcing these INDs leads to the addition of 60 tuples. Thus, we obtain an IADB for ***movies*** including 434 tuples, i.e. *only 0.6% of the total number of tuples in* ***movies***, keeping exactly the same functional and inclusion dependencies.

6. Conclusion

In this paper, IADBs have been defined as small subsets of existing databases, but satisfying exactly the same set of functional and inclusion dependencies. The existence of IADBs with a bounded size has been pointed out by a theorem, in the case of non-circular set of INDs. An algorithm is proposed to compute such "small" IADBs, whose complexity is linear in the number of generator sets and the size of positive and negative borders of INDs. An implementation of our algorithm has been performed, taking in account the case of circular INDs; results on a real-life database show the feasibility of our approach. Moreover, the size of the constructed IADB represents 0.6% of the size of the initial database with exactly the same set of satisfied and non-satisfied FDs and INDs. Due to this reduction, IADBs can be seen as a *semantic sampling* of existing databases.

Although Armstrong databases as such yield limited benefits, IADBs are likely to show better results for databases maintenance or for KDD processing. Indeed, they convey both data semantics (FDs and INDs) and the intended meaning of tuples from the underlying databases. Moreover, since *null values* are quite common in real life databases, IADBs may also have a lot of null values. Therefore, IADBs give also a faithful representation of the initial DB with respect to null values, in addition to a faithful representation of the dependencies. This is indeed the case in the movies database.

We conclude the paper by pointing out three open problems: given a database **d**, what is the size of "a smallest" IADB for **d**? How far is this size from the bounds given in Theorem 1? Is the associated decision problem NP-complete? At the moment, we obtain one interesting result, in the case of FDs alone: we successfully formulated the problem as an integer programming problem, and thus solved it for ***movies*** using CPlex 7.1 [33]. We were able to compute the size of the smallest IADB. Results suggest two remarks: (1) the optimum is very far from the lower bound of Theorem 1 and (2) the optimum is very close to the size obtained by our program (which implement a simple heuristic). As a consequence we have good reasons to think that the lower bound of Theorem 1 can be improved, taking into account not only the number of generator sets, but also their structure.

References

[1] M. Casanova, R. Fagin, C. Papadimitriou, Inclusion dependencies and their interaction with functional dependencies, J. Comput. Syst. Sci. 24 (1) (1984) 29–59.
[2] E.F. Codd, Recent investigations in relational data base systems, in: J.L. Rosenfeld (Ed.), International Federation for Information Processing (IFIP'74), Stockholm, Sweden, 1974, pp. 1017–1021.
[3] M. Levene, M.W. Vincent, Justification for inclusion dependency normal form, IEEE Trans. Knowledge Data Eng. 12 (2) (2000) 281–291.
[4] M.A. Casanova, L. Tucherman, A.L. Furtado, Enforcing inclusion dependencies and referencial integrity, in: F. Bancilhon, D.J. DeWitt (Eds.), International Conference on Very Large Data Bases, (VLDB'88), Los Angeles, CA, USA, Morgan Kaufmann, Los Altos, 1988, pp. 38–49.
[5] H. Mannila, K.-J. Räihä, Inclusion dependencies in database design, in: International Conference on Data Engineering, (ICDE'86), Los Angeles, CA, USA, IEEE Computer Society, Silver Spring, MD, 1986, pp. 713–718.
[6] R.J. Miller, M.A. Hernández, L.M. Haas, L. Yan, C.T.H. Ho, R. Fagin, L. Popa, The clio project: managing heterogeneity, ACM Sigmod Rec. 30 (1) (2001) 78–83.
[7] J. Gryz, Query folding with inclusion dependencies, in: International Conference on Data Engineering, (ICDE'98), Orlando, FL, USA, IEEE Computer Society, 1998, pp. 126–133.
[8] Q. Cheng, J. Gryz, F. Koo, T.Y.C. Leung, L. Liu, X. Qian, B. Schiefer, Implementation of two semantic query optimization techniques in DB2 universal database, in: M.P. Atkinson, M.E. Orlowska, P. Valduriez, S.B. Zdonik, M.L. Brodie (Eds.), International Conference on Very Large Data Bases, (VLDB'99), Edinburgh, Scotland, UK, Morgan Kaufmann, Los Altos, 1999, pp. 687–698.
[9] W.W. Armstrong, Dependency structures of database relationships, in: J.L. Rosenfeld (Ed.), International Federation for Information Processing, (IFIP'74), Stockholm, Sweden, 1974, pp. 580–583.
[10] R. Fagin, M. Vardi, Armstrong databases for functional and inclusion dependencies, Inform. Process. Lett. 16 (1983) 13–19.

[11] A.M. Silva, M.A. Melkanoff, A method for helping discover the dependencies of a relation, in: H. Gallaire, J.-M. Nicolas, J. Minker (Eds.), Advances in Data Base Theory, (ADBT'81), Toulouse, France, 1979, pp. 115–133.
[12] H. Mannila, K.-J. Räihä, Design by example: an application of Armstrong relations, J. Comput. Syst. Sci. 63 (2) (1986) 126–141.
[13] Y. Huhtala, J. Karkkainen, P. Porkka, H. Toivonen, TANE: an efficient algorithm for discovering functional and approximate dependencies, Comput. J. 42 (2) (1999) 100–111.
[14] N. Novelli, R. Cicchetti, Functional and embedded dependency inference: a data mining point of view, Inform. Syst. 26 (7) (2001) 477–506.
[15] S. Lopes, J.-M. Petit, L. Lakhal, Functional and approximate dependencies mining: databases and FCA point of view, JETAI 14 (2/3) (2002) 93–114 (special issue).
[16] F. De Marchi, J.-M. Petit, Zigzag: a new algorithm for discovering large inclusion dependencies in relational databases, in: International Conference on Data Mining, (ICDM'03), Melbourne, FL, USA, IEEE Computer Society, Silver Spring, MD, 2003, pp. 27–34.
[17] C. Beeri, M. Dowd, R. Fagin, R. Statman, On the structure of Armstrong relations for functional dependencies, J. ACM 31 (1) (1984) 30–56.
[19] F. De Marchi, S. Lopes, J.-M. Petit, Samples for understanding data-semantics in relations, in: M.-S. Hacid, Z.W. Ras, D.A. Zighed, Y. Kodratoff (Eds.), International Symposium on Methodologies for Intelligent Systems (ISMIS'02), Lyon, France, Lecture Notes in Artificial Intelligence, vol. 2366, Springer, Berlin, 2002, pp. 565–573.
[21] M. Albrecht, E. Buchholz, A. Düsterhöft, B. Thalheim, An informal and efficient approach for obtaining semantic constraints using sample data and natural language processing, in: L. Libkin, B. Thalheim (Eds.), Semantics in Databases, Lecture Notes in Computer Science, vol. 1358, Springer, Berlin, 1995, pp. 1–28.
[22] F. De Marchi, S. Lopes, J.-M. Petit, F. Toumani, Analysis of existing databases at the logical level: the DBA companion project, ACM Sigmod Rec. 32 (1) (2003) 47–52.
[23] S. Lopes, F. De Marchi, J.-M. Petit, DBA Companion: a tool for logical database tuning, in: Demo Session of International Conference on Data Engineering (ICDE'04), IEEE Computer Society, Silver Spring, MD, 2004, http://www.isima.fr/ ~demarchi/dbacomp/.
[24] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
[25] M. Levene, G. Loizou, A Guided Tour of Relational Databases and Beyond, Springer, Berlin, 1999.
[26] W.W. Armstrong, C. Delobel, Decomposition of functional dependencies in relations, ACM Trans. Database Syst. 5 (4) (1980) 404–430.
[27] H. Mannila, H. Toivonen, Levelwise search and borders of theories in knowledge discovery, Data Min. Knowledge Discovery 1 (1) (1997) 241–258.
[28] F. De Marchi, S. Lopes, J.-M. Petit, Efficient algorithms for mining inclusion dependencies, in: C.S. Jensen, K.G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, M. Jarke (Eds.), International Conference on Extending Database Technology (EDBT'02), Prague, Czech Republic, Lecture Notes in Computer Science, vol. 2287, Springer, Berlin, 2002, pp. 464–476.
[29] M. Levene, G. Loizou, How to prevent interaction of functional and inclusion dependencies, Inform. Process. Lett. 71 (3–4) (1999) 115–125.
[30] S.D. Bay, The UCI KDD archive, Technical Report, University of California, Irvine, CA, Department of Information and Computer Science, 1999, [http://kdd.ics.uci.edu].
[31] J. Demetrovics, V. Thi, Some remarks on generating Armstrong and inferring functional dependencies relation, Acta Cybern. 12 (2) (1995) 167–180.
[32] M. Kantola, H. Mannila, K.J. Räihä, H. Siirtola, Discovering functional and inclusion dependencies in relational databases, Int. J. Intell. Syst. 7 (1992) 591–607.
[33] F. De Marchi, Découverte et visualisation par l'exemple des dépendances fonctionnelles et d'inclusion dans les bases de données relationnelles, Ph.D. Thesis, 2003 (in French), http://www.isima.fr/~demarchi/these_book.ps.