# Efficient Discovery of Functional Dependencies and Armstrong Relations

Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal

Laboratoire LIMOS, Université Blaise Pascal - Clermont-Ferrand II
Campus Universitaire des Cézeaux
24 avenue des Landais
63177 Aubière cedex, France
{slopes,jmpetit,llakhal}@libd2.univ-bpclermont.fr

**Abstract** In this paper, we propose a new efficient algorithm called Dep-Miner for discovering minimal non-trivial functional dependencies from large databases. Based on theoretical foundations, our approach combines the discovery of functional dependencies along with the construction of *real-world Armstrong relations* (without additional execution time). These relations are small Armstrong relations taking their values in the initial relation. Discovering both minimal functional dependencies and real-world Armstrong relations facilitate the tasks of database administrators when maintaining and analyzing existing databases. We evaluate Dep-Miner performances by using a new benchmark database. Experimental results show both the efficiency of our approach compared to the best current algorithm (i.e. Tane), and the usefulness of real-world Armstrong relations.

## 1 Introduction and Motivation

Functional dependencies, introduced in [11], are by far the most common integrity constraints in the real world [26,21]. They are very important when designing or analyzing relational databases. Discovering functional dependencies hidden in a database has been addressed by various approaches, among which we quote [24,31,25,18].

Armstrong relations, introduced in [16], are closely related to functional dependencies: such relations exactly satisfy a set of functional dependencies. They can show both the existence and the nonexistence of functional dependencies for a given relation [15,24,25]. Algorithms for computing Armstrong relations from functional dependencies are given in [6,24,14,12]. In this paper, we introduce the concept of *real-world Armstrong relations*. Such relations are small Armstrong relations only populated with values actual from the initial relation.

Discovering both minimal functional dependencies and real-world Armstrong relations could greatly facilitate the tasks of database administrators (DBA) when maintaining existing databases and reorganizing their schemas. We call such a reorganization logical tuning: for instance, the DBA could assess relevance of discovered functional dependencies by using small relations sampling the initial

C. Zaniolo et al. (Eds.): EDBT 2000, LNCS 1777, pp. 350–364, 2000.

relations, and once these dependencies are proved to be useful, he can perform relation normalization. The motivation behind normalization is to remove the problems that are caused by the update anomalies and redundancies [26,21].

For addressing the problem of discovering minimal non-trivial functional dependencies, a theoretical framework is proposed in [25,26]. The underlying approach is based on the concept of agree set [6]. This set groups all the attributes having the very same values for a given couple of tuples. From agree sets, maximal sets[1] are derived, and from maximal sets, all minimal non-trivial functional dependencies can be generated.

In this paper, we propose a new efficient algorithm called Dep-Miner for discovering agree sets, maximal sets, left-hand sides (LHS) of minimal non-trivial functional dependencies and real-world Armstrong relations. Our approach is defined under the assumption of limited main memory resources and its feasibility does not depend on the volume of handled data. Since database accesses are only performed during the computation of agree sets, Dep-Miner takes in input a small representation of a relation, called *stripped partition databases* derived from [13,32,18]. From them, new characterizations of agree sets are given. These characterizations show that stripped partition databases are informationaly equivalent to relations in our context and provide efficient algorithms for discovering agree sets from large relations. From agree sets, a characterization of maximal sets is introduced. Then, a levelwise algorithm[2] is proposed for computing the LHS of minimal non-trivial functional dependencies. It is based on the characterization of LHS as the set of minimal transversals of a simple hypergraph [25,26]. An existence condition for real-world Armstrong relation is given as well as the algorithm for generating such a relation.

Evaluations of Dep-Miner performances are achieved by using a new benchmark database. Experimental results show both the efficiency of the approach compared to the best current algorithm (i.e. Tane [18]), and the usefulness of real-world Armstrong relations. Indeed, we observed that these relations were small sizes and thus form a good sampling of the initial relation.

*Paper organization.* In section 2, some definitions and results in relational database theory are presented. Our approach is detailed in section 3 and two versions of the algorithm Dep-Miner are presented. In section 4, we explain how to achieve Armstrong relations with the algorithms Dep-Miner. Section 5 details experimental results and section 6 concludes the paper by giving further research work.

## 2 Basic Definitions

This section is devoted to setting the groundwork of our approach. It briefly resumes definitions and results from relational database theory, which are relevant in our context [26,3,21].

[1] Also called *intersection generators* in [6] or *meet-irreductible sets* in [17,14].

[2] This kind of algorithm has been extensively used in data mining [4,29,27].

Let $R$ be a *relation schema*. If $X \subseteq R$ and $t$ is a tuple, we denote by $t[X]$ the restriction of $t$ to $X$.

A *functional dependency* over $R$ is an expression $X \rightarrow A$ where $X \subseteq R$ and $A \in R$. The functional dependency $X \rightarrow A$ *holds* in a relation $r$ (denoted by $r \models X \rightarrow A$) if and only if $\forall t_i, t_j \in r$, $t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A]$. A functional dependency $X \rightarrow A$ is *minimal* if $A$ is not functionally dependent on any proper subset of $X$. The functional dependency $X \rightarrow A$ is *trivial* if $A \in X$. We denote by *dep*$(r)$ the set of all functional dependencies that hold in $r$: $dep(r) = \{X \rightarrow A / X \cup A \subseteq R, r \models X \rightarrow A\}$. Let $F$ and $G$ be two sets of functional dependencies, $F$ is a *cover* of $G$ if $F \models G$ (this notation means that each dependency $f \in G$ holds in any relation satisfying all the dependencies in $F$) and $G \models F$.

For complementing previous definitions, agree sets, maximal sets and LHS sets are introduced.

Let $t_i$ and $t_j$ be tuples and $X$ an attribute set. The tuples $t_i$ and $t_j$ *agree* on $X$ if $t_i[X] = t_j[X]$. The *agree set* of $t_i$ and $t_j$ is defined as follows: $ag(t_i, t_j) = \{A \in R / t_i[A] = t_j[A]\}$. If $r$ is a relation, $ag(r) = \{ag(t_i, t_j) / t_i, t_j \in r,\ t_i \neq t_j\}$.

A maximal set is an attribute set $X$ which, for some attribute $A$, is the largest possible set not determining $A$. We denote by $max(dep(r), A)$ the set of maximal sets for $A$ w.r.t. *dep*$(r)$:
$max(dep(r), A) = \{X \subseteq R / r \not\models X \rightarrow A \text{ and } \forall Y \subseteq R, X \subset Y, r \models Y \rightarrow A\}$; and $MAX(dep(r)) = \bigcup_{A \in R} max(dep(r), A)$.

From maximal sets, functional dependencies can be inferred as follows [25]:

The set of LHS of functional dependencies w.r.t. *dep*$(r)$ and an attribute $A$ is denoted by $lhs(dep(r), A)$: $lhs(dep(r), A) = \{X \subseteq R / r \models X \rightarrow A \text{ and } \forall X' \subset X, r \not\models X' \rightarrow A\}$. The set $\{X \rightarrow A / X \in lhs(dep(r), A), A \in R\}$ is a cover of *dep*$(r)$.

For finding LHS of functional dependencies from maximal sets, the notion of hypergraph is to be introduced. A collection $\mathcal{H}$ of subsets of $R$ is a *simple hypergraph* if $\forall X \in \mathcal{H}, X \neq \emptyset$ and $(X, Y \in \mathcal{H}$ and $X \subseteq Y \Rightarrow X = Y)$ [8]. Elements of $\mathcal{H}$ are called the *edges* of the hypergraph and elements of $R$ are the *vertices* of the hypergraph. The collection $cmax(dep(r), A)$ of complements of maximal sets $max(dep(r), A)$ is a simple hypergraph. A *transversal* T of $\mathcal{H}$ is a subset of $R$ intersecting all the edges of $\mathcal{H}$, i.e. $T \cap E \neq \emptyset, \forall E \in \mathcal{H}$. A *minimal transversal* of $\mathcal{H}$ is a transversal T such that it does not exist a transversal $T'$, $T' \subset T$. The collection of minimal transversals of $\mathcal{H}$ is denoted by $\mathrm{Tr}(\mathcal{H})$. Minimal transversals of simple hypergraph are related to LHS of functional dependencies: $\mathrm{Tr}(cmax(dep(r), A)) = lhs(dep(r), A)$.

## 3 Dep-Miner Algorithm

Our approach is depicted in figure 1: from the initial relation, a stripped partition database is extracted; using such partitions, agree sets are computed; and thus, maximal sets are generated. On the one hand, they are used to build Armstrong relations. On the other hand, deriving their complements is straightforward and

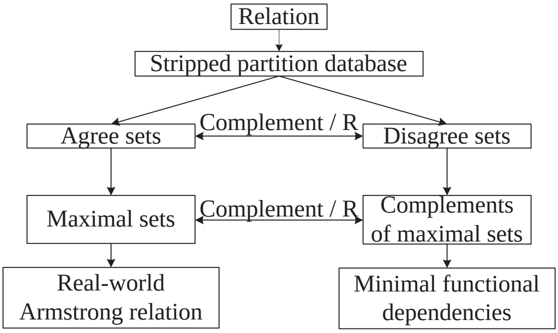
Relation
Stripped partition database
Agree sets
Complement / R
Disagree sets
Maximal sets
Complement / R
Complements of maximal sets
Real-world Armstrong relation
Minimal functional dependencies

**Fig. 1.** General framework

then LHS of functional dependencies are computed. Let us notice that approaches presented in [24,19,25,26] fit in this general framework without necessarily covering all the presented steps. Moreover, they operate by loading the dealt data in main memory without a special emphasis on the computation of agree sets. Algorithm 1 (see below) presents the different steps of Dep-Miner.

**Algorithm 1.** Dep-Miner: Discovering minimal functional dependencies and real-world Armstrong relations

```
Input: a relation $r$
Output: minimal functional dependencies and real-world Armstrong relation for $r$
1: AGREE_SET: computes agree sets from $r$
2: CMAX_SET: derives complements of maximal sets from agree sets
3: LEFT_HAND_SIDE: computes LHS of functional dependencies from complements
   of maximal sets
4: FD_OUTPUT: outputs functional dependencies
5: ARMSTRONG_RELATION: builds real-world Armstrong relation from maximal
   sets and $R$
```

### 3.1 Finding Agree Sets

A naive algorithm for computing agree sets in a relation $r$ works as follows: for each couple of tuples $(t_i, t_j)$ in $r$, compute $ag(t_i, t_j)$ as defined in the previous section. If $p$ is the number of tuples in the relation and $n$ is the number of attributes, the time complexity of this algorithm is in $O(np^2)$. When $p$ is large, the algorithm becomes impractical due to the number of couples (plus the overhead due to the cost of $ag(t_i, t_j)$).

We propose a new approach to compute agree sets which aims to decrease the number of candidate couples. For meeting such needs, we reduce the initial relation using the concept of *stripped partition database* and new characterizations of agree sets are proposed in order to minimize the number of couples. Furthermore, an interesting aspect would be to avoid the cost of $ag(t_i, t_j)$.

From stripped partition databases, two algorithms are proposed: the former implements the new approach to compute agree sets; the latter provides an optimization of the previous algorithm which is more efficient when handling large relations.

**Stripped partition databases.** The fundamental idea underlying our approach is to provide a reduced representation of a relation. This can be achieved using the notion of partitions [13,32,18].

*Partitions.* Two tuples $t_i$ and $t_j$ are *equivalent* with respect to a given attribute set $X$ if $t_i[A] = t_j[A] \forall A \in X$. The *equivalence class* of a tuple $t_i \in r$ with respect to a given set $X \subseteq R$ is defined by $[t_i]_X = \{t_j \in r / t_i[A] = t_j[A], \forall A \in X\}$. The set $\pi_X = \{[t]_X / t \in r\}$ of equivalence classes is a *partition* of $r$ under $X$. In the sequel, we use a positive integer unique to $t$ as an identifier for each tuple $t$.

*Example 1.* Let us consider the following relation representing the assignment of employees to departments.

| Tuple No. | empnum | depnum | year | depname | mgr |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 85 | Biochemistry | 5 |
| 2 | 1 | 5 | 94 | Admission | 12 |
| 3 | 2 | 2 | 92 | Computer Sce | 2 |
| 4 | 3 | 2 | 98 | Computer Sce | 2 |
| 5 | 4 | 3 | 98 | Geophysics | 2 |
| 6 | 5 | 1 | 75 | Biochemistry | 5 |
| 7 | 6 | 5 | 88 | Admission | 12 |

For briefness, attributes empnum, depnum, year, depname, mgr are renamed $A$, $B$, $C$, $D$, $E$ respectively. The partition associated to attribute $A$ is: $\pi_A = \{\{1,2\},\{3\},\{4\},\{5\},\{6\},\{7\}\}$.

*Stripped partitions.* Such partitions group equivalence classes having a size greater than one. In fact, when an equivalence class encompasses a single element, the associated tuple does not share the values of the considered attribute set with any other tuple in the relation. The stripped partition for an attribute set $X$ is defined by: $\widehat{\pi_X} = \{c \in \pi_X / |c| > 1\}$

*Example 2.* The stripped partition for attribute $A$ is achieved by removing equivalence classes of size one: $\widehat{\pi_A} = \{\{1,2\}\}$.

*Stripped partition databases.* The new representation of a relation is called a stripped partition database. It encompasses stripped partitions for each attribute. Let $r$ be a relation over $R$. A *stripped partition database* $\widehat{r}$ of $r$ is defined as follows: $\widehat{r} = \bigcup_{A \in R} \widehat{\pi_A}$.

Computing stripped partition database from a relation is straightforward (it would correspond to the pre-processing phase in a data mining context).

**Characterizing agree sets.** We firstly need to define the set $MC$ of maximal equivalence classes induced by a stripped partition database.

*Maximal equivalence classes.* Let $\widehat{r}$ be a stripped partition database. The set $MC$ of maximal equivalence classes of $\widehat{r}$ is defined as follows:
$MC = Max_{\subseteq}\{c \in \widehat{\pi} / \widehat{\pi} \in \widehat{r}\}$.

*Example 3.* Continuing our example, the set of maximal equivalence classes is the following:
$MC = \{\{1, 2\}, \{1, 6\}, \{2, 7\}, \{3, 4, 5\}\}$.

For building agree sets, we only consider couples of tuples belonging to a common equivalence class of $MC$ (because tuples in two different equivalence classes disagree for each attribute of $R$). This results from the lemma 1 which proves the correctness of algorithm 2 presented below.

**Lemma 1.** *[22] Let $r$ be a relation. $ag(r) = \bigcup_{c \in MC} ag(c)$.*

**The first algorithm.** The first proposed algorithm (see below algorithm 2) results from the lemma 1. It operates as follows: The first step (line 1) computes the maximal equivalence classes from a stripped partition database. Then, for each maximal equivalence class, all possible couples of tuples are generated (lines 4 to 7). Corresponding agree sets are then computed (lines 8 to 12): an attribute is added to the agree set of two tuples if these tuples are in a common equivalence class in the stripped partition for this attribute. Finally, the set of agree sets is updated (lines 13 to 24).

**Algorithm 2.** AGREE_SET: Computes agree sets from stripped partition databases

**Input:** the stripped partition database $\widehat{r}$ of a relation $r$
**Output:** the agree sets of $r$: $ag(r)$

```
 1: $MC := Max_{\subseteq}\{c \in \widehat{\pi_A} / \widehat{\pi_A} \in \widehat{r}\}$
 2: $ag(r) := \emptyset$
 3: $couples := \emptyset$
 4: for all maximal equivalence classes $c \in MC$ do
 5:    for all couple $(t, t') \in c$ do
 6:       $couples := couples \cup (t, t')$
 7:       $ag(t, t') := \emptyset$
 8: for all $\widehat{\pi_A} \in \widehat{r}$ do
 9:    for all equivalence class $c \in \widehat{\pi_A}$ do
10:       for all $(t, t') \in couples$ do
11:          if $t \in c$ and $t' \in c$ then
12:             $ag(t, t') := ag(t, t') \cup A$
13: for all couple $(t, t') \in couples$ do
14:    $ag(r) := ag(r) \cup ag(t, t')$
```

*Example 4.* From the set $MC$, the generated couples are:
$\{(1, 2), (1, 6), (2, 7), (3, 4), (3, 5), (4, 5)\}$.
This algorithm discovers the following agree sets: $ag(r) = \{\emptyset, A, BDE, CE, E\}$.

Compared with the naive algorithm, the number of couples is reduced and the cost of $ag(t, t')$ is avoided. However, the proposed algorithm requires storing all couples that can generate agree sets. Since the number of these couples can be very great, we cannot assume that they always fit into main memory. The solution used to avoid this problem is computing agree sets as soon as a fixed number of couples was generated. More precisely, when a threshold (associated to the number of tuples) is reached, corresponding agree sets are computed from the current set of couples. This set is then deleted and the process continues by examining the remaining couples.

However, the computation can be time consuming and the algorithm becomes less efficient when the number of couples is great, i.e. when equivalence classes are large or when they are numerous. We propose therefore another characterization of agree sets which originates to a new algorithm more efficient in such a case.

**Another characterization of agree sets.** The fundamental idea under this new characterization of agree sets is to preserve, for each tuple, the identifiers of equivalence classes in which the considered tuple appears. Then, computing the agree set of two tuples can be merely performed by achieving the intersection of their identifier set, and getting the associated attributes.

Let us assume that $\widehat{\pi_A} = \{\widehat{\pi_{A,0}}, \dots, \widehat{\pi_{A,k}}\}$. We denote by $ec(t)$ the set of identifiers of equivalence classes in which the tuple $t$ appears: $ec(t) = \{(A, i)/A \in R \text{ and } t \in \widehat{\pi_{A,i}}\}$

*Example 5.* In our example, for attribute $E$, $\widehat{\pi_E} = \{\widehat{\pi_{E,0}}, \widehat{\pi_{E,1}}, \widehat{\pi_{E,2}}\}$, where $\widehat{\pi_{E,0}} = \{1, 6\}, \widehat{\pi_{E,1}} = \{2, 7\}, \widehat{\pi_{E,2}} = \{3, 4, 5\}$
For the second tuple, the indentifier set is $ec(2) = \{(A, 0), (B, 1), (D, 1), (E, 1)\}$

We can now give a characterization of agree sets.

**Lemma 2.** *[22] Let $t_i$ and $t_j$ be two tuples. $ag(t_i, t_j) = \{A \in R/\exists k \text{ s.t. } (A, k) \in ec(t_i) \cap ec(t_j)\}$.*

*Example 6.* Let us consider $ec(1) = \{(A, 0), (B, 0), (D, 0), (E, 0)\}$ and
$ec(2) = \{(A, 0), (B, 1), (D, 1), (E, 1)\}$. Since $ec(1) \cap ec(2) = \{(A, 0)\}$,
$ag(1, 2) = A$.

**The second algorithm.** From lemma 2, we propose a second algorithm for exhibiting agree sets (see below algorithm 3). The first step (lines 2 to 5) states the relationship between tuples and equivalence classes: for each tuple in the stripped partition database, the equivalence classes in which the considered tuple appears are preserved (line 5). In the second step (lines 6 to 9), agree sets are computed: for each couple in maximal equivalence classes, the agree set of the couple is computed from the relationships previously stated (line 9).

### 3.2 Finding Maximal Sets

For exhibiting maximal sets from agree sets, we introduce a new characterization[3] of the set of maximal sets for the attribute A: $max(dep(r), A)$.

[3] In [25], a similar result is used for yielding complements of maximal sets from complements of agree sets (disagree sets). However, it is not explicitly stated contrarily to Lemma 3.

**Algorithm 3.** AGREE_SET 2: Computes agree sets from stripped partition databases

```
Input: the stripped partition database $\widehat{r}$ of a relation $r$
Output: the agree sets of $r$: $ag(r)$
$ag(r) := \emptyset$
for all $\widehat{\pi_A} \in \widehat{r}$ do
    for all equivalence class $\widehat{\pi_{A,i}} \in \widehat{\pi_A}$ do
        for all tuple $t \in \widehat{\pi_{A,i}}$ do
            $ec(t) := ec(t) \cup (A, i)$
$MC := Max_{\subseteq}\{c \in \widehat{\pi_A} / \widehat{\pi_A} \in \widehat{r}\}$
for all maximal equivalence classes $c \in MC$ do
    for all couple $(t, t') \in c$ do
        $ag(r) := ag(r) \cup \{A \in R / \exists j$ s.t. $(A, j) \in ec(t) \cap ec(t')\}$
```

**Lemma 3.** *[22]* $max(dep(r), A) = Max_{\subseteq}\{X \in ag(r) / A \notin X,\ X \neq \emptyset\}$.

As mentioned in section 2, we need to compute complement of maximal sets for achieving LHS of minimal functional dependencies. Algorithm 4 yields complements of maximal sets from agree sets. Its correctness results from lemma 3.

Firstly, we compute maximal sets for each attribute in $R$ (lines 1 to 2): for an attribute $A$ in $R$, agree sets which do not contain $A$ and which are maximal with respect to inclusion are added to the set of maximal sets (line 2). Finding the complement of maximal sets (lines 3 to 6) is straightforward.

**Algorithm 4.** CMAX_SET: Computes complement of maximal sets

```
Input: the agree sets over $r$: $ag(r)$
Output: complements of maximal sets: $CMAX(dep(r))$
1: for all attributes $A \in R$ do
2:     $max(dep(r), A) := Max_{\subseteq}\{X \in ag(r) / A \notin X\}$
3: for all attributes $A \in R$ do
4:     $cmax(dep(r), A) := \emptyset$
5:     for all $X \in max(dep(r), A)$ do
6:         $cmax(dep(r), A) := cmax(dep(r), A) \cup (R \setminus X)$
```

*Example 7.* When applied to our example, the previous algorithm yields the following results for attribute $A$:
$max(dep(r), A) = \{BDE, CE\}$ and $cmax(dep(r), A) = \{AC, ABD\}$.

### 3.3 Finding Left-Hand Sides of Functional Dependencies

Minimal transversals of the simple hypergraph $cmax(dep(r), A)$ provide LHS of minimal functional dependencies (see section 2). We propose a new levelwise algorithm (see algorithm 5) for computing minimal transversals of a simple hypergraph.

The set $L_i$ of candidate sets of size i is initialized with attributes appearing in *cmax*$(dep(r), A)$. The collection of minimal transversals is computed (from lines 4 to 8): for each set l in $L_i$, we test if l is a transversal (line 5). In this case, l is saved (line 5) in $LHS_i$ of LHS of minimal functional dependencies of size i and removed (line 6) from $L_i$ (all supersets of l are non minimal transversals). Next level is generated (line 7) by adapting the *Apriori-gen* function [4].

**Algorithm 5.** LEFT_HAND_SIDE: Computes LHS of minimal functional dependencies
**Input:** complements of maximal sets: $CMAX(dep(r))$
**Output:** the LHS of minimal functional dependencies: $lhs(dep(r))$

```
1: for all attributes $A \in R$ do
2:    $i := 1$
3:    $L_i := \{B/B \in X, X \in cmax(dep(r), A)\}$
4:    while $L_i \neq \emptyset$ do
5:       $LHS_i[A] := \{l \in L_i / l \cap X \neq \emptyset, \forall X \in cmax(dep(r), A)\}$
6:       $L_i := L_i \setminus LHS_i[A]$
7:       $L_{i+1} := \{l'/|l'| = i + 1$ and $\forall l \subset l'/|l| = i, l \in L_i\}$
8:       i := i + 1
9:    $lhs(dep(r), A) := \bigcup_i LHS_i[A]$
```

*Example 8.* We obtain the following sets:
$lhs(dep(r), A) = \{A, BC, CD\}$, $lhs(dep(r), B) = \{AC, AE, B, D\}$,
$lhs(dep(r), C) = \{AB, AD, AE, C\}$, $lhs(dep(r), D) = \{AC, AE, B, D\}$,
$lhs(dep(r), E) = \{B, C, D, E\}$.

These set leads to the following minimal funtional dependencies:

$r \models BC \rightarrow A$
$r \models CD \rightarrow A$
$r \models AC \rightarrow B$
$r \models AE \rightarrow B$
$r \models D \rightarrow B$
$r \models AB \rightarrow C$
$r \models AD \rightarrow C$
$r \models AE \rightarrow C$
$r \models AC \rightarrow D$
$r \models AE \rightarrow D$
$r \models B \rightarrow D$
$r \models B \rightarrow E$
$r \models C \rightarrow E$
$r \models D \rightarrow E$

## 4 Generating Real-World Armstrong Relations

Exhibiting functional dependencies could yield a huge amount of results and taking advantages of them is far from trivial. Generally, all the functional dependences cannot be taken into account to normalize the relational schema. In fact, only some inferred functional dependencies are relevant when modifying the database structure [26]. Two reasons justify that:

1. Some functional dependencies could accidentally hold in a relation extension which represents the state of the data at a given time. There is no guarantee for the validity of these dependencies in another relation extension.
2. Functional dependencies can express two things [7,28]: either an association of attributes which represents relevant information which is interesting to preserve, or just an integrity constraint between the data.

For making decision of discarding a functional dependency or not, possible alternatives are:

1. requesting the DBA to make such a decision;
2. using clues given by a workload of SQL statement for example by studying duplicate attribute sequences [23];
3. providing some help to the DBA for example with a sample of the initial relation.

The next step of our approach fits in the latter trend.

Let us notice that a rather similar issue is also addressed by recent data mining approaches because discovered knowledge could be so voluminous that it could not be directly used [20,5,30]. Nevertheless, we do not provide a comparison between these approaches and ours because proposed solutions widely differ.

An algorithm to construct an Armstrong relation from maximal sets is proposed in [6,24]. Let us assume that $C = \{X_0, \ldots, X_n\}$ where $X_0 = R$ and $X_i \in MAX(dep(r))$. Each $X_i \in C$ is associated with the tuple $t_i$ defined as follows:

$$t_i[A] = \begin{cases} 0 & \text{if } A \in X_i, \\ \mathrm{i} & \text{if } A \notin X_i. \end{cases} \tag{1}$$

The relation $r = \{t_0, \ldots, t_n\}$ is an Armstrong relation of size $|MAX(dep(r))|+1$.

Under similar assumptions, real-world Arsmtrong relations are built up from an initial relation. We firstly present what we mean by real-world Armstrong relation:

Informally, a real-world Armstrong relation is an Armstrong relation satisfying the three following properties:

1. it is an equivalent representation of the initial relation as regards functional dependencies;
2. its values are taken among those of the initial relation;
3. its size is often smaller of several orders of magnitude than the size of the initial relation.

**Definition 1.** *Let $r$ be a relation over $R$. A real-world Armstrong relation $\overline{r}$ over $R$ is defined as follows:*

1. *$\overline{r}$ is an Armstrong relation satisfying $dep(r)$;*
2. *$|\overline{r}| = |MAX(dep(r))| + 1$;*
3. *$\forall A \in R, \forall t_i \in \overline{r}, t_i[A] \in \pi_A(r)$ where $\pi_A(r)$ is the projection of $r$ on $A$.*

The existence of a real-world Armstrong relation depends on the number of distinct values for each attribute in the relation $r$: That leads to the following result.

**Proposition 1.** *[22] Let $r$ be a relation over $R$. A real-world Armstrong relation $\overline{r}$ over $R$ exists if and only if $\forall A \in R$, $|\pi_A(r)| \geq |\{X \in MAX(dep(r))/A \notin X\}| + 1$.*

This condition means that, in the initial relation, each attribute must necessarily have enough different values in order to construct real-world Armstrong relations. Under this condition, we can build them as follows:

Suppose $C = \{X_0, \dots, X_n\}$ where $X_0 = R$ and $X_i \in MAX(dep(r))$. For each $X_i \in C$, associate the tuple $t_i$ such that: $\forall A \in R, \pi_A(r) = \{v_{A0}, \dots, v_{Ak}\}$

$$t_i[A] = \begin{cases} v_{A0} & \text{if } A \in X_i, \\ v_{Ai} & \text{if } A \notin X_i. \end{cases} \tag{2}$$

*Example 9.* From our example, the following Armstrong relation and real-world Armstrong relation can be generated from $MAX(dep(r)) \cup R$

| empnum | depnum | year | depname | mgr | empnum | depnum | year | depname | mgr |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 85 | Biochemistry | 5 |
| 0 | 1 | 1 | 1 | 1 | 1 | 5 | 94 | Admission | 12 |
| 2 | 0 | 2 | 0 | 0 | 3 | 1 | 92 | Biochemistry | 5 |
| 3 | 3 | 0 | 3 | 0 | 4 | 2 | 85 | Geophysics | 5 |

Let us underline that real-world Armstrong relation is more informative than the other one. As shown in the next section, their sizes can be significantly smaller than the size of the initial relation.

## 5 Performances

To test the performances of our algorithms, we performed several experiments on an Intel Pentium II with a CPU clock rate of 350 Mhz, 256 MB of main memory and running Windows NT 4. We implemented the algorithms using the C++ language and STL (Standard Template Library). Attribute sets are implemented as bit vectors to provide set operations in constant time. The DBMS accesses are done by ODBC to remain independent of the DBMS. We used two DBMSs during the tests: Oracle and MS Access.

Firstly, we give an overview of the Tane algorithm against which we compare the performances of Dep-Miner. Then, we present the new benchmark database used for the tests and show the obtained results.

### 5.1 The Tane Algorithm

Several algorithms for discovering functional dependencies have been presented [24,31,25]. However, the Tane algorithm [18] is the best current algorithm for the discovery of minimal non-trivial functional dependencies. Moreover, Tane can also provide approximate functional dependencies. It partitions the set of tuples of a relation according to their attribute values. Thus it preserves the information about which tuples agree on a set of attributes. To check if a functional dependency holds, it verifies whether the tuples agree on the right-hand side whenever they agree on the left-hand side. The approach is based on a levelwise algorithm [27]. Functional dependencies are searched starting with dependencies having small left-hand side (i.e. from dependencies that are not likely satisfied). It prunes the search space as soon as possible.

For the tests, due to the limitation of the downloadable version of Tane (available at [2]) to relations with less than 32 attributes and the fact that Tane is implemented in C under Linux, we have implemented our version of Tane in order to compare it with Dep-Miner.

### 5.2 The Benchmark Database

We generated synthetic data sets (i.e. relations) in order to control various parameters during the tests. By this way, the pros and cons for the two algorithms can be studied in more depth.

We firstly create a table with $|R|$ attributes in the database and then insert $|r|$ tuples. Each inserted value depends on the parameter $c$ which is the rate of identical values. It controls the number of identical values in a column of the table. For example, if $c$ has a value of 50% for an attribute and the number of tuples is 1000, this means that each value for this attribute is chosen between 500 possible values.

### 5.3 Experiments with Synthetic Data

In this section, we present experimental results obtained with generated data. Tests were made on various relations classified in three groups: data sets without constraints, data sets with parameter $c$ set to 30% and data sets with parameter $c$ fixed to 50%. Due to the lack of space, only the second dataset will be presented here (see figure 2). The full set of tests (result tables and figures) can be found in [22].

The number of attributes varies from 10 to 60 and the number of tuples from 10,000 to 100,000. The execution times (in seconds) are shown in figure 2,

In these tests, we compare two versions of Dep-Miner to Tane. The former (called Dep-Miner) implements algorithm 2 for computing agree sets. The latter (called Dep-Miner 2) implements algorithm 3 to perform the very same task.

For discovering functional dependencies, Dep-Miner is faster than Tane in all cases. The difference grows along with the number of attributes. Dep-Miner 2 is more efficient than Tane when the number of attributes or the number of tuples are large.

For Armstrong relations, we observe that their size is small compared with the size of the original relations. Most of the times, the number of tuples in generated real-world Armstrong relations varies from $1/100$ to $1/10,000$ compared with the number of tuples of the original relations.

## 6 Conclusion

In this paper, we propose a new approach intended for a twofold objective: discovering minimal non-trivial functional dependencies holding in a given relation;

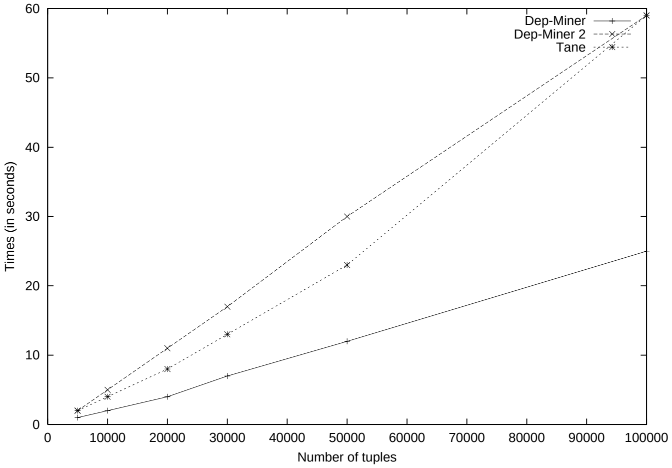

**Fig. 2.** Execution times (in seconds) for correlated data (30%)

achieving a real-world Armstrong relation, which can be seen as a loss-less sample of the initial relation.

The approach fits in a theoretical framework proposed in [24,25,26] for addressing the very same issue. Nevertheless, it differs from related work because we put the emphasis on the efficiency of the discovery of functional dependencies and real-world Armstrong relations. In this context, new solutions are proposed by using techniques originated by data mining. Each step of the approach is provided with formal foundations ensuring the correctness of the underlying algorithms.

The main benefit of our twofold discovery approach is that the DBA is provided with two different representations. On one hand functional dependencies could be used for normalizing existing relation schemas. On the other hand, real-world Armstrong relations are particularly useful for better understanding relation schemas, and aiding to select only relevant functional dependencies among the whole (and possibly voluminous) set of extracted dependencies.

*Perspectives for Database Administration.* Reducing database administration functions is recognized as being a new challenge in database community. In this context, the aim of the so-called "plug and play databases" is facilating the database administrator tasks and dealing with information discovery [9]. For example, in the context of the *AutoAdmin* project [1], physical database design is investigated for tuning index definitions in order to improve performances of the system [10].

In a similar way, existing logical database constraints should be fully understood. Providing the DBA with such a knowledge is particularly critical not only for improving application performances but also for guaranteeing data consistency. We believe that promising applications of the presented work fit in such a research direction.

**Acknowledgments.** We thank Rosine Cicchetti for her help during the writing of this paper.

## References

1. Autoadmin Project, Microsoft research, database group, `http://www.research.microsoft.com/db`.
2. WWW page `http://www.cs.helsinki.fi/research/fdk/datamining/tane`.
3. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1995.
4. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago de Chile, Chile*, pages 487–499, 1994.
5. Roberto Bayardo and Rakesh Agrawal. Mining the most interesting rules. In *Proceedings of the Fifth International Conference on Knowledge Discovery & Data Mining, San Diego, CA, USA*, 1999.
6. Catriel Beeri, Martin Dowd, Ronald Fagin, and Richard Statman. On the structure of Armstrong relations for functional dependencies. *Journal of the ACM*, 31(1):30–46, 1984.
7. Catriel Beeri and Michael Kifer. An integrated approach to logical design of relational database schemes. *ACM Transaction on Database Systems*, 11(2):134–158, 1986.
8. Claude Berge. *Graphs and Hypergraphs*. North-Holland Mathematical Library 6. American Elsevie 1976, 2d rev. ed. edition, 1976.
9. Philip A. Bernstein, Michael L. Brodie, Stefano Ceri, David J. DeWitt, Michael J. Franklin, Hector Garcia-Molina, Jim Gray, Gerald Held, Joseph M. Hellerstein, H. V. Jagadish, Michael Lesk, David Maier, Jeffrey F. Naughton, Hamid Pirahesh, Michael Stonebraker, and Jeffrey D. Ullman. The Asilomar report on database research. *SIGMOD Record*, 27(4):74–80, 1998.
10. Surajit Chaudhuri and Vivek R. Narasayya. Autoadmin 'what-if' index analysis utility. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA*, pages 367–378, 1998.
11. E. F. Codd. Further normalization of the data base relational model. Technical Report 909, IBM Research, 1971.
12. Ethan Collopy and Mark Levene. Evolving example relations to satisfy functional dependencies. In *Proceedings of the International Workshop on Issues and Applications of Database Technology*, pages 440–447, 1998.
13. Stavros S. Cosmadakis, Paris C. Kanellakis, and Nicolas Spyratos. Partition semantics for relations. *Journal of Computer and System Sciences*, 33(2):203–233, 1986.
14. János Demetrovics, Leonid Libkin, and Ilya B. Muchnik. Functional dependencies in relational databases: A lattice point of view. *Discrete Applied Mathematics*, 40:155–185, 1992.
15. Ronald Fagin. Armstrong databases. Technical Report 5, IBM Research Laboratory, 1982.
16. Ronald Fagin. Horn clauses and database dependencies. *Journal of the ACM*, 29(4):952–985, 1982.
17. Georg Gottlob and Leonid Libkin. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica*, 9(4):385–402, 1990.

18. Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings of the Fourteenth IEEE International Conference on Data Engineering*, pages 392–401, 1998.
19. Martti Kantola, Heikki Mannila, Kari-Jouko Räihä, and Harri Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7:591–607, 1992.
20. Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management, Gaithersburg, Maryland*, pages 401–407, 1994.
21. Mark Levene and Georges Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-verlag London Limited, 1999.
22. Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and armstrong relations (complete version) `http://libd2.univ-bpclermont.fr/publications`. Technical report, LIMOS, 1999.
23. Stéphane Lopes, Jean-Marc Petit, and Farouk Toumani. Discovery of "interesting" data dependencies from a workload of SQL statements (poster). In Jan M. Zytkow and Jan Rauch, editors, *Proceedings of the Principles of Data Mining and Knowledge Discovery, Prague, Czech Republic*, volume 1704, pages 430–435, 1999.
24. Heikki Mannila and Kari-Jouko Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33(2):126–141, 1986.
25. Heikki Mannila and Kari-Jouko Räihä. Algorithms for inferring functional dependencies from relations. *Data and Knowledge Engineering*, 12(1):83–99, 1994.
26. Heikki Mannila and Kari-Jouko Räihä. *The Design of Relational Databases*. Addison Wesley, 1994.
27. Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
28. V.M. Markowitz and J.A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, 1990.
29. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the Seventh International Conference on Database Theory, Jerusalem, Israël*, pages 398–416, 1999.
30. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Mining bases for association rules using galois closed sets (poster). In *Proceedings of the Sixteenth IEEE International Conference on Data Engineering, February 29 - March 3, San Diego, CA, USA*. IEEE Computer Society, 2000.
31. Iztok Savnik and Peter A. Flach. Bottom-up induction of functional dependencies from relations. In *Proceedings of the AAAI-93Workshop on Knowledge Discovery in Databases*, pages 174–185, 1993.
32. Nicolas Spyratos. The partition model: A deductive database model. *ACM Transaction on Database Systems*, 12(1):1–37, 1987.