# An incremental approach for maintaining functional dependencies

Ghada Gasmi, Lotfi Lakhal, Yahya Slimani

Intelligent Data Analysis 16 (2012) 365–381
DOI 10.3233/IDA-2012-0529
IOS Press


# An incremental approach for maintaining functional dependencies

Ghada Gasmi[a,*], Lotfi Lakhal[b] and Yahya Slimani[a]
[a]*Computer Science Department, Faculty of Sciences of Tunis, University of Tunis El Manar, Tunisia*
[b]*Laboratoire d'Informatique Fondamentale de Marseille, Aix-Marseille, Université IUT d'Aix en Provence, France*

**Abstract.** A general assumption in all existing algorithms permitting to mine functional dependencies is that the database is static. However, real life databases are frequently updated. To the best of our knowledge, the discovery of functional dependencies in dynamic databases has never been studied. A naïve solution consists in re-applying one of the existing algorithms to discover functional dependencies holding on the updated database. Nevertheless, in many domains, where response time is crucial, re-executing algorithms from scratch would be inacceptable. In this paper, we propose a new technique that makes use of the previously discovered results to cut down the amount of work that has been done to discover the new set of functional dependencies satisfied by the updated database.

Keywords: Functional dependencies, evolving databases, data mining, knowledge discover, maintenance

## 1. Introduction

Across a wide variety of fields, data is collected and accumulated at a dramatic pace. The traditional method of extracting knowledge from data relies on manual analysis and interpretation. Nevertheless, this method is completely impractical in many domains. Hence, there is an urgent need for tools to assist humans in extracting knowledge from the rapidly growing volumes of digital data. These tools are the subject of the data mining field. Association mining is an important task in data mining, which consists in taking data as input and provides associations, such as association rules, implications, or functional dependencies as output. In this paper, we focus on functional dependencies.

Originally, the study of functional dependencies (FDs) has been motivated by the fact that they could express constraints holding on a relation independently of a particular instance [1]. Later, Mannila and Räihä studied FDs with a data mining point of view. Indeed, the idea was introduced in [18] as the inference of functional dependencies problem. Its principle consists in determining a cover of all functional dependencies holding on a given relation $r$. Motivations for addressing functional dependencies inference arise in several areas [3,6,12,15,16,19,21,23,24]. Indeed, FDs were applied in database management [9,17], data reverse engineering [22], query optimization [20], data streams [10] and data mining [12].

*Corresponding author: Ghada Gasmi, Computer Science Department, Faculty of Sciences of Tunis, University of Tunis El Manar, Tunisia. E-mail: ghada.gasmi@gmail.com.

A crucial study of the dedicated literature allows one to note that a general assumption in all existing algorithms is that the database is static. However, real life databases are dynamic where they are constantly updated. Hence, a possible solution consists in re-applying one of the existing algorithms on the updated database. This solution though simple, has disadvantages. All the previous computation done to discover FDs is wasted and the process of FDs discovery must restart from the scratch. Nevertheless, the more the size of the database and the frequency of its update increase the more this solution becomes time consuming and inacceptable in many applications. In dynamic databases, the problem of functional dependencies inference is reformulated as follows: determining a cover of all functional dependencies holding on the updated relation with lower costs.

In this paper, we show that the problem of FDs inference in dynamic database can be reduced to a problem of maintaining FDs. Indeed, given a relation $r$ and a new tuple $t$, we propose to discover the canonical cover of the FDs holding on $(r \cup t)$ by taking advantages of the canonical cover of FDs holding on $r$. For that:

1. we start by verifying that the updated relation does not violate any FD of the canonical cover of FDs holding on $r$;
2. if there are some FDs violated by the updated relation, we maintain partially the canonical cover of FDs holding on $r$ in order to deduce the canonical cover of FDs holding on $(r \cup t)$.

The main idea of our proposal is described graphically by Fig. 1. We conducted experiments on synthetic databases in order to assess our approach and compare it against re-applying an existing algorithm on the updated database. The experimental results showed that our approach is 23 to 24125 times faster than re-executing existing algorithm from the scratch.

The remainder of the paper is organized as follows. Section 2 gives some formal definitions and notations used throughout the paper. Section 3 presents the problem of maintaining FDs and describes our proposal: the INCFDS algorithm. The empirical study about the utility of our algorithm is provided in Section 4. The paper ends with a conclusion of our contributions and sketches forthcoming issues in the Section 5.

## 2. Theoretical background

In this section, we present the basic concepts used throughout the paper.

### 2.1. Functional dependencies

In what follows, we briefly review definitions and results from relational database theory [1,11].

Let $\mathcal{A} = \{a_1, \ldots, a_m\}$ be a finite set of attributes. Each attribute $a_i$ has a finite domain, denoted $dom(a_i)$, representing the values that $a_i$ can take on. For a subset $X = \{a_i, \ldots, a_j\}$ of $\mathcal{A}$, $dom(X)$ is the Cartesian product of the domains of the individual attributes in $X$. A relation $r$ on $\mathcal{A}$ is a finite set of tuples $\{t_1, \ldots, t_n\}$ from $\mathcal{A}$ to $dom(\mathcal{A})$ with the restriction that for each tuple $t \in r$, $t[X]$ must be in $dom(X)$, such that $X \subseteq \mathcal{A}$ and $t[X]$ denotes the restriction of the tuple $t$ to $X$.

**Definition 1.** *Let $r$ be a relation on $\mathcal{A}$. A functional dependency (FD) over $\mathcal{A}$ is an expression $X \rightarrow A$ where $X \subset \mathcal{A}$ and $A \in \mathcal{A}$. We refer to $X$ as the antecedent and $A$ as the consequent. A FD $X \rightarrow A$ holds on $r$ (denoted $r \models X \rightarrow A$) 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 FD $X \rightarrow A$ is minimal (or elementary) if and only if $r \models X \rightarrow A$ and $\forall Z \subset X$, $r \not\models Z \rightarrow A$. We denote by $\mathcal{F}_r$ the set of all functional dependencies satisfied by $r$.*

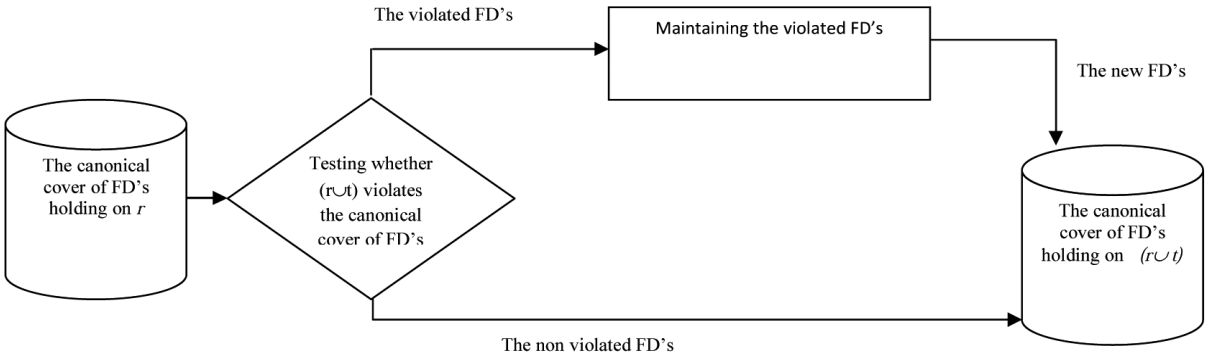


Fig. 1. The outline of the proposed approach.

**Example 1.** *Let us consider the following relation the relation $r$ describing a hotel rooms. For briefness, attributes "HotelID", "NumRoom", "TypeRoom", "CatHoltel" and "Price" are renamed $A, B, C, D$ and $E$, respectively.*

*$BD \rightarrow A$ is a functional dependency satisfied by $r$. It is minimal since it does not exist a subset $Z$ of $BD$ such that $r \models Z \rightarrow A$. The functional dependency $A \rightarrow B$ is not satisfied by $r$, since for the tuples $(t_1, t_3) \in r$, we have $t_1[A] = t_3[A]$ however $t_1[B] \neq t_3[B]$.*

**Definition 2.** *Let $\mathcal{F}$ and $\mathcal{G}$ be two sets of functional dependencies. $\mathcal{F}$ is a cover of $\mathcal{G}$ if $\mathcal{F} \models \mathcal{G}$ (this notation means that each dependency of $\mathcal{G}$ holds in any relation satisfying all the dependencies in $\mathcal{F}$) and $\mathcal{G} \models \mathcal{F}$.*

**Definition 3.** *Let $r$ be a relation on $\mathcal{A}$. The canonical cover of $\mathcal{F}_r$ is defined as follows: $Cover(\mathcal{F}_r) = \{X \rightarrow A | X \subset \mathcal{A}, A \in \mathcal{A}, r \models X \rightarrow A, X \rightarrow A \; is \; minimal\}$.*

**Definition 4.** *Let $X \subseteq \mathcal{A}$ and $Cover(\mathcal{F}_r)$ be the canonical cover of $\mathcal{F}_r$. The closure of $X$ w.r.t $Cover(\mathcal{F}_r)$, denoted $(X_{Cover(\mathcal{F}_r)})^+$, is given by $X \cup \{A | Y \rightarrow A \in Cover(\mathcal{F}_r), Y \subseteq X\}$.*

### *2.2. Hypergraph theory*

In this section, we start by recalling some formal definitions and the necessary properties on hypergraphs. Then, we illustrate the connection between functional dependency mining and hypergraph theory. For more theoretical issues the reader is referred to [4].

A hypergraph $\mathcal{H}$ is an ordered pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of a finite set $\mathcal{V} = \{V_1, \ldots, V_n\}$ and a family $\mathcal{E} = \{E_1, \ldots, E_m\}$ of subsets of $\mathcal{V}$. The elements of $\mathcal{V}$ are called nodes while the elements of $\mathcal{E}$ are called hyperedges of the hypergraph $\mathcal{H}$. $\mathcal{H}$ is simple if for every pair $E_i, E_j \in \mathcal{E}$, $E_j \subset E_i \Rightarrow j = i$. $Min(\mathcal{H})$ is the set of minimal hyperedges of $\mathcal{H}$ with respect to set inclusion, i.e., $Min(\mathcal{H}) = \{E_i \in \mathcal{E} | \nexists E_j \in \mathcal{E} \; such \; that \; E_j \subset E_i\}$.

**Example 2.** *Let us consider the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V} = \{A, B, C, D, E\}$ and $\mathcal{E} = \{AB, BDE, BE, ABCD\}$. Then, $\mathcal{H}$ is not simple since $AB \subset ABCD$. $Min(\mathcal{H}) = \{AB, BE\}$.*

Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A set $\mathcal{T} \subseteq \mathcal{V}$ is called a transversal (or, *hitting set*) of $\mathcal{H}$ if it intersects all the hyperedges of $\mathcal{H}$, *i.e.,* $\mathcal{T} \cap E_i \neq \emptyset, \forall E_i \in \mathcal{E}$. A transversal $\mathcal{T}$ is minimal if no proper subset $\mathcal{T}'$

of $\mathcal{T}$ is a transversal of $\mathcal{H}$. The family of all minimal transversals of $\mathcal{H}$ constitutes a simple hypergraph called the transversal hypergraph of $\mathcal{H}$ denoted by $Tr(\mathcal{H})$. It has $\mathcal{V}$ as nodes and the minimal transversals of $\mathcal{H}$ as hyperedges.

The following lemmas capture important relations between a hypergraph and its transversal hypergraph.

**Lemma 1.** $Tr(\mathcal{H}) = Tr(Min(\mathcal{H}))$.

**Lemma 2.** $Tr(E_i) = \{\{e\}|\{e\} \in E_i\}$.

In [4], the author propose to compute $Tr(H)$ incrementally. Its idea is formally presented as follows.

**Lemma 3.** *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph such that $\mathcal{E} = \{E_1, \ldots, E_m\}$. Let us define a sequence $\mathcal{T}_0, \ldots, \mathcal{T}_m$ by $\mathcal{T}_0 = \{\emptyset\}$ and $\mathcal{T}_i = Min(T \cup \{e\}|T \in \mathcal{T}_{i-1}, \{e\} \in E_i)$. Then, $\mathcal{T}_j = Tr(\{E_1, \ldots, E_j\}), j \in [0 \ldots m]$. Thus, $\mathcal{T}_m = Tr(\mathcal{H})$.*

**Example 3.** *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph such that $\mathcal{E} = \{AB, BCD\}$. $\mathcal{T}_0 = \{\emptyset\}$, $\mathcal{T}_1 = Min(A, B)$, $\mathcal{T}_2 = Min(AB, AC, AD, B, BC, BD)$. Thus, $\mathcal{T}_2 = \{B, AD, AC\} = Tr(\mathcal{H})$.*

Hypergraph theory is an important subfield of discrete mathematics with many relevant applications in both theoretical and applied computer science [7,8] including distributed systems, databases [5,11], boolean circuits [13] and artificial intelligence [14]. Furthermore, it is important to note that a connexion between computing transversal hypergraph and canonical cover of FDs satisfied by a relation $r$ was studied in [11]. Indeed, it was pointed out that determining minimal FDs having $A$ as consequent is reduced to compute transversal hypergraph. Moreover, we will show later that hypergraph theory will be of use to mine FDs in evolving databases.

## 3. Incremental updating of functional dependencies

The problem of functional dependency mining consists in discovering a cover for the functional dependencies holding on a relation $r$. The majority of the dedicated works discover the canonical cover since it is unique for a given relation $r$. Nevertheless, one general assumption in all existing approaches is that the database is static [3,6,12,15,16,19,21,23,24]. However, real life databases are dynamic and they are constantly updated. A possible solution consists in re-applying one of the existing algorithms on the updated database. This solution though simple, has disadvantages. All the previous computation done to discover FDs is wasted and the process of FDs discovery must restart from the scratch. Nevertheless, the more the size of the database and the frequency of its update increase the more this solution becomes time consuming and inacceptable in many applications. Hereafter, we propose the first algorithm, called IncFDs, which takes into consideration the dynamic feature of the databases. The idea behind our algorithm is "natural". Indeed, when some changes occur on a given situation, we, first, ask ourselves if the characteristics of the initial situation are still valid. After, if there are some characteristics that are no longer valid, we modify them in order to make them sound on the new situation. Let us consider our initial relation $r$ "characterized" by the canonical cover of the FDs holding on $r$. After "changing" $r$ by augmenting it with $t$:

1. we verify whether $Cover(\mathcal{F}_r)$ could be also be the canonical cover of the FDs holding on $(r \cup t)$.
2. if there are some minimal FDs of $Cover(\mathcal{F}_r)$ that are violated by $(r \cup t)$, we update them in order to deduce $Cover(\mathcal{F}_{r \cup t})$.

Table 1
Example of a relation

| Tuple ID | HotelID | NumRoom | TypeRoom | CatHoltel | Price |
|---|---|---|---|---|---|
| $t_1$ | 1 | 100 | 1 | 2 | 50 |
| $t_2$ | 4 | 101 | 1 | 2 | 50 |
| $t_3$ | 1 | 102 | 2 | 2 | 70 |
| $t_4$ | 1 | 200 | 1 | 2 | 50 |
| $t_5$ | 2 | 101 | 3 | 3 | 100 |
| $t_6$ | 2 | 200 | 1 | 3 | 70 |
| $t_7$ | 1 | 100 | 3 | 2 | 50 |

### *3.1. Checking whether $Cover(\mathcal{F}_r) = Cover(\mathcal{F}_{r \cup t})$*

In order to check whether $Cover(\mathcal{F}_r)$ is equal to $Cover(\mathcal{F}_{r \cup t})$, a solution consists in verifying that the updated relation $(r \cup t)$ does not violate any minimal FD of $Cover(\mathcal{F}_r)$. It means that for each minimal functional dependency $X \rightarrow A$, we have to verify, for each couple $(t, t'), t' \in r$ and $t$ is the tuple to insert into $r$, that if $t[X] = t'[X] \Rightarrow t[A] = t'[A]$. Obviously, the more the sizes of $Cover(\mathcal{F}_r)$ and the relation $r$ increase the more disadvantageous this solution would be. To offset this shortcoming, we have to reduce: (i) the number of candidate couples; (ii) the number of minimal FDs that should be checked (*i.e.,* checked whether they are not violated by $(r \cup t)$).

#### *3.1.1. Reducing the candidate couples*

We note that tuples of $r$ that must be considered are those ones sharing with $t$ the same value of at least one attribute. It would be interesting then to restrict $r$ to these tuples. For that, we introduce the notion of equivalence classes of $r$ with respect to $t$ defined as follows:

**Definition 5.** *Let $r$ be a relation on $\mathcal{A}$ and $t$ be a tuple. Then, for a given attribute $A \in \mathcal{A}$, the equivalence class of $r$ with respect to $t$ is $r(A)_t = \{t' | t[A] = t'[A], t' \in r\}$. We denote by $EC(r)_t = \{r(A)_t | A \in \mathcal{A}\}$ all equivalence classes of $r$ with respect to $t$.*

**Example 4.** *Let us consider the relation of Table 1 and suppose that it contains only the six first tuples. The equivalence classes of $r$ with respect to $t_7$ are:*
$r(A)_{t_7} = \{t_1, t_3, t_4\}$;
$r(B)_{t_7} = \{t_1\}$;
$r(C)_{t_7} = \{t_5\}$;
$r(D)_{t_7} = \{t_1, t_2, t_3, t_4\}$;
$r(E)_{t_7} = \{t_1, t_2, t_4\}$.
*Then, $EC(r)_{t_7} = \{r(A)_{t_7}, r(B)_{t_7}, r(C)_{t_7}, r(D)_{t_7}, r(E)_{t_7}\}$.*
*Through this example, we show that thanks to the equivalence classes of $r$ with respect to $t_7$, we reduced the number of candidate couples. As we remark, the couple $(t_6, t_7)$ was not considered.*

#### *3.1.2. Minimizing the number of minimal FDs to check*

Initially, all minimal FDs of $Cover(\mathcal{F}_r)$ are candidate to be violated by the updated relation $(r \cup t)$. Nevertheless, it would be interesting to narrow the search space of the minimal FDs that are candidate to be violated. A minimal FD $X \rightarrow A$ of $Cover(\mathcal{F}_r)$ still holds on $(r \cup t)$ if and only if $\forall (t, t')$ such that $t' \in r$, if $t[X] = t'[X] \Rightarrow t[A] = t'[A]$. In other words, if $t$ and $t'$ are agree on $X$, they have to be agree also on $A$. Hence, we are sure that for each minimal FD $X \rightarrow A$ of $Cover(\mathcal{F}_r)$, if $A$ belongs to all attribute sets on which $(t, t')$ are agree, then $X \rightarrow A$ still holds on $(r \cup t)$. Thus, computing attribute

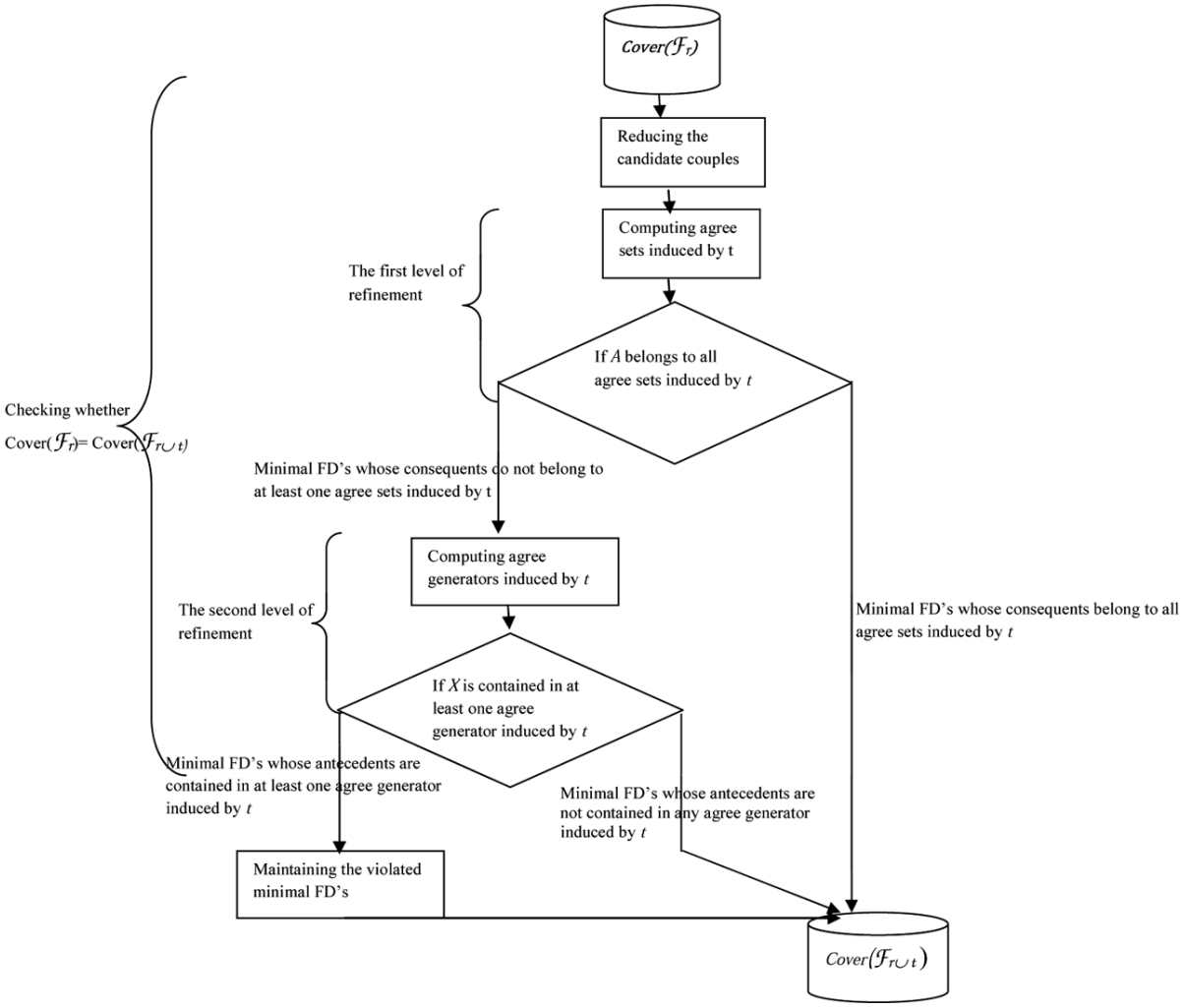


Fig. 2. The refinement of the search space of the minimal FDs candidate to be violated.

sets on which $(t, t')$ are agree, such that $t' \in r$ (*i.e., agree sets induced by t*) would guide us to identify a first set of minimal FDs of $Cover(\mathcal{F}_r)$ which are satisfied also by $(r \cup t)$. Consequently, a first level of refinement of the search space of the candidate violated FDs is based on the agree sets induced by $t$.

For a minimal FD $X \rightarrow A$, which belongs to the remaining set of the candidate violated FDs of $Cover(\mathcal{F}_r)$, we have at least one agree set induced by $t$ that does not contain $A$. In addition, two cases can arise:

- if $X$ is not contained in any agree set induced by $t$, then $X \rightarrow A$ is not violated by $(r \cup t)$;
- if $X$ is contained in at least one agree set induced by $t$, then $X \rightarrow A$ is violated by $(r \cup t)$.

Thus, identifying agree sets induced by *t* which contain at least one antecedent of the remaining minimal FDs, candidate to be violated, (*i.e., agree generators induced by t*) will allow us to determine the set of minimal FDs that have to be updated.

Through Fig. 2, we describe graphically how we reduce the search space of the minimal FDs that are candidate to be violated. Hereafter, we will detail the both levels of refinement summarized above.

*3.1.2.1. A first level of refinement based on the agree sets induced by t*

As said before, computing agree sets induced by *t* would offer us a key permitting to reduce the search space of the minimal FDs candidate to be violated.

*Computing the agree sets induced by t*: Before giving the formal definition of agree sets induced by *t*, we first recall the definition of an agree set. According to a couple of tuples, an agree set is defined as the set of all attributes which have the same values for the considered tuples. Formally, this concept was defined in [15] as follows:

**Definition 6.** *Let t and t' be two tuples of r and $X \subseteq \mathcal{A}$ be a set of attributes. Then, the tuples t and $t'$ are agree on X if and only if $t[X] = t'[X]$. Hence, according to t and $t'$, the agree set, denoted $Ag(t,t') = \{A | t[A] = t'[A], A \in \mathcal{A}\}$. $Ag(r) = \{Ag(t,t') | (t,t') \in r, t \neq t'\}$ denotes all agree sets of r.*

Based on Definition 6, we define formally the agree sets induced by a tuple $t$ as follows:

**Definition 7.** *Let r be a relation and t be a tuple. Then, with respect to r, agree sets induced by the tuple t are $\{Ag(t,t') | t' \in r\}$ and they are denoted by $Ag(r)_t$.*

For computing agree sets induced by a tuple $t$, a naïve solution consists in computing $Ag(t,t')$ for each couple of tuples $(t,t')$ such that $t' \in r$ and $t$ the tuple to insert into $r$. Nevertheless, the more the number of tuples of $r$ increases the more expensive this solution would be. To palliate this problem, we provide a new characterization of the agree sets induced by $t$ based on the equivalence classes of $r$ with respect to $t$. Before providing the new characterization of the agree sets induced by $t$, we need to define the maximal equivalence classes of $r$ with respect to $t$.

**Definition 8.** *Let r be a relation and t be a tuple. The maximal classes of r with respect to t are: $MC(r)_t = Max\{r(A)_t | r(A)_t \in EC(r)_t\}$*

**Example 5.** *For the equivalence classes of r with respect to $t_7$ (c.f., Example 4). The maximal classes are: $MC(r)_{t_7} = \{\{t_5\}, \{t_1, t_2, t_3, t_4\}\}$.*

Through Proposition 1, we point out that agree sets induced by $t$ can be computed straightforwardly from the equivalence classes of $r$ with respect to $t$.

**Proposition 1.** *Let r be a relation and t be a tuple to insert into r. The agree sets induced by t are given by: $Ag(r)_t = \{Ag(t,t') | t' \in c, c \in MC(r)_t\}$ Such that $Ag(t,t') = \{A | r(A)_t\}$.*

*Proof. We recall that $Ag(r)_t$ contains all agree sets induced by t: $Ag(r)_t = \{Ag(t,t') | t' \in r\}$ (c.f., Definition 7). Since the equivalence classes of r with respect to t contain the tuples of r sharing with t the same value of at least one attribute, then agree sets induced by t can be written as follows: $Ag(r)_t = \{Ag(t,t') | t' \in c, c \in EC(r)_t\}$. However, a tuple $t'$ can belong to more than one equivalence class. Then, agree sets induced by t can be written as follows: $Ag(r)_t = \{Ag(t,t') | t' \in c, c \in MC(r)_t\}$.*

*Now, let us prove that $Ag(t,t') = \{A | t' \in c, c \in EC(r)_t, A \in \mathcal{A}\}$. According to Definition 6, $Ag(t,t') = \{A | t[A] = t'[A], A \in \mathcal{A}\}$. This means that the agree set of the couple $(t,t')$ is the maximal set of attributes whose values are shared by t and $t'$. Hence, according to Definition 5, agree set of the couple $(t,t')$ can be written as follows: $Ag(t,t') = \{A | t' \in c, c \in EC(r)_t, A \in \mathcal{A}\}$.*

**Example 6.** *Let us consider the relation of Table 1 and suppose that it contains only the six first tuples and we want insert the tuple $t_7$. The agree sets induced by $t_7$ are computed as follows. The tuples of $r$ that have to be considered are deduced from the maximal equivalence classes of $r$ with respect to $t$ (c.f., Example 5). They are equal to: $\{t_1, t_2, t_3, t_4, t_5\}$. After, we can compute straightforwardly agree sets induced by the tuple $t_7$ as follows.*
*$Ag(t_7, t_1) = ABDE$ because $t_1$ belongs to $r(A)_{t_7}, r(B)_{t_7}, r(D)_{t_7}$ and $r(E)t_7$.*
*$Ag(t_7, t_2) = DE$ because $t_2$ belongs to $r(D)_{t_7}$ and $r(E)_{t_7}$*
*$Ag(t_7, t_3) = AD$ because $t_3$ belongs to $r(A)_{t_7}$ and $r(D)_{t_7}$.*
*$Ag(t_7, t_4) = ADE$ because $t_4$ belongs to $r(A)_{t_7}, r(D)_{t_7}$ and $r(E)_{t_7}$.*
*$Ag(t_7, t_5) = C$ because $t_5$ belongs to $r(C)_{t_7}$.*
*Consequently, $Ag(r)_{t_7} = \{ABDE, AD, ADE, DE, C\}$.*

*Pruning a first set of candidate violated FDs*: Once agree sets induced by $t$ are computed, we can determine a first set of FDs which are not violated after the insertion of $t$. Indeed, we are sure that each minimal FD $X \rightarrow A$ of $Cover(\mathcal{F}_r)$, such that $A$ belongs to all agree sets induced by $t$, still holds on $(r \cup t)$.

**Example 7.** *Let us consider the relation $r$ of Table 1 and suppose that it contains only the three first tuples and we will insert tuple $t_4$. The canonical cover of $r$ is the following:*

| | | | | |
|---|---|---|---|---|
| $B \rightarrow A$ | $AC \rightarrow B$ | $B \rightarrow C$ | $\emptyset \rightarrow D$ | $B \rightarrow E$ |
| | $AE \rightarrow B$ | $E \rightarrow C$ | | $C \rightarrow E$ |

*After computing the agree sets induced by $t_4$, we obtain $Ag(r)_{t_4} = \{ACDE, CDE, AD\}$. Hence, we are sure that minimal FDs having $D$ as consequent will be valid after the insertion of $t_4$ (since $D \in ACDE, D \in CDE$ and $D \in AD$). Thus, we narrow the search space of the candidate violated FDs by pruning those having $D$ as consequent.*

*3.1.2.2. A second level of refinement based on agree generators induced by $t$*

As said before (*c.f.*, Sub-section 3.1.2), computing agree generators induced by $t$ allows us to determine the set of minimal FDs that have to be updated (*i.e.,* those violated after the insertion of the new tuple).

*Identifying the agree generators induced by $t$*: Appellation of agree generators induced by $t$ is justified by the fact that we generate a new canonical cover of $(r \cup t)$ if we have at least one agree generator induced by $t$. Formally, agree generators induced by $t$ are defined as follows.

**Definition 9.** *An agree set $X$ induced by $t$, is said to be an agree generator induced by $t$ if and only if $X \neq (X_{Cover(\mathcal{F}_r)})^+$. The whole set of agree generators induced by $t$ is denoted by $Gen(r)_t$.*

**Example 8.** *Let us consider the relation $r$ of Table 1 and suppose that it contains only the three first tuples and we will insert tuple $t_4$. The canonical cover of $r$ is given in Example 7. After computing the agree sets induced by $t_4$, we obtain $Ag(r)_{t_4} = \{ACDE, CDE, AD\}$. Let us identify the agree generators induced by $t_4$.*
*For $ACDE$, $(ACDE_{Cover(\mathcal{F}_r)})^+ = ABCDE$. Then, $ACDE$ is an agree generator induced by $t_4$.*
*For $CDE$, $(CDE_{Cover(\mathcal{F}_r)})^+ = CDE$. Then, $CDE$ is not an agree generator induced by $t_4$.*
*For $AD$, $(AD_{Cover(\mathcal{F}_r)})^+ = AD$. Then, $AD$ is not an agree generator induced by $t_4$.*
*Consequently, $Gen(r)_{t_4} = \{ACDE\}$.*

*Determining the set of violated FDs*: Proposition 2 ensures that, thanks to agree generators induced by $t$, we can determine the set of FDs which will be violated after the insertion of $t$.

**Proposition 2.** *$X \rightarrow A$ is violated by $(r \cup t)$ if and only if $\exists G \in Gen(r)_t$ such that $X \subseteq G$ and $A \notin G$.*

*Proof. For a minimal FD $X \rightarrow A$, which belongs to the remaining set of the candidate violated FDs of $Cover(\mathcal{F}_r)$, we have at least one agree set induced by $t$ that does not contain $A$. We denote by $Ag(r)_t/A$ the collection of agree sets induced by $t$ that do not contain $A$. Then, two cases can arise:*

1. *if $X$ is not contained in any agree set of $Ag(r)_t/A$, then $X \rightarrow A$ is not violated by $(r \cup t)$.*
2. *if $X$ is contained in at least one agree set of $Ag(r)_t/A$, then $X \rightarrow A$ is violated by $(r \cup t)$. Hence, for each agree set $Y$ of $Ag(r)_t/A$ that contains $X$, $A$ belongs necessarily to the closure of $Y$ (c.f., Definition 4) and according to Definition 9, $Y$ is an agree generator induced by $t$.*

**Example 9.** *Let us continue with the Example 8. We recall that after the first refinement the remaining set of the candidate violated FDs of $Cover(\mathcal{F}_r)$ are:*

| | | | |
|---|---|---|---|
| $B \rightarrow A$ | $AC \rightarrow B$ | $B \rightarrow C$ | $B \rightarrow E$ |
| | $AE \rightarrow B$ | $E \rightarrow C$ | $C \rightarrow E$ |

*According to Example 8, $Gen(r)_{t_4} = \{ACDE\}$. Then, the violated functional dependencies are those whose antecedents are contained in $ACDE$ and their consequents do not belong to $ACDE$: $AC \rightarrow B$ and $AE \rightarrow B$. Thus, we see that thanks to the second refinement, we pruned five other FDs.*

### *3.2. Maintaining the set of violated FDs*

If there are FDs that are violated after the insertion of $t$, means that $Cover(\mathcal{F}_r) \neq Cover(\mathcal{F}_{r \cup t})$. Hence, we have to compute $Cover(\mathcal{F}_{r \cup t})$. However, we have already computed a part of this canonical cover. Indeed, $Cover(\mathcal{F}_{r \cup t})$ contains:

- the minimal FDs of $Cover(\mathcal{F}_r)$ that are not violated after the insertion of $t$. These FDs are already determined in the previous step.
- the minimal FDs which replace the violated ones.

Hereafter, we provide the following properties related to the agree generators induced by $t$ which will be of use to comprehend the method used to maintain the violated FDs.

**Property 1.** *Let $r$ be a relation and $X$ be an agree set induced by $t$. If $\exists (t', t") \in r$ such that $Ag(t', t") = X$, then, $X$ is not an agree generator induced by $t$.*

*Proof. Suppose that $\exists (t', t") \in r$ such that $Ag(t', t") = X$. Hence, it $\nexists Y \rightarrow A \in Cover(\mathcal{F}_r)$ such that $Y \subseteq X$ and $A \notin X$ (because this means that $t'$ and $t"$ are agree on $Y$ however they are not agree on $A$). Hence, $X = (X_{Cover(\mathcal{F}_r)})^+$. Consequently, $X$ cannot be an agree generator induced by $t$.*

Thanks to Property 1, we can deduce the following property.

**Property 2.** *Let $Ag(r \cup t)$ and $Ag(r)$ be the agree sets of, respectively, the updated relation$(r \cup t)$ and $r$. Then, $Ag(r \cup t) = Ag(r) \cup Gen(r)_t$.*

*Proof. It is obvious that* $Ag(r \cup t) = Ag(r) \cup Ag(r)_t$. *Since it may exist an agree set* $X$ *induced by* $t$ *such that* $Ag(t', t") = X$ *and* $(t', t") \in r$. *Then, we can deduce that* $Ag(r \cup t) = Ag(r) \cup Gen(r)_t$.

Now, let us explain the idea of maintaining the violated FDs. A functional dependency $X \rightarrow A \in Cover(\mathcal{F}_r)$ is violated by $(r \cup t)$ means that:

- $r \models X \rightarrow A$: $\nexists Y \in Ag(r)$ such that $X \subseteq Y$ *and* $A \notin Y$.
- $(r \cup t) \not\models X \rightarrow A$: $\exists G \in Gen(r)_t$ such that $X \subseteq G$ and $A \in G$ (*c.f.,* Proposition 2).

Hence, when we update a violated minimal functional dependency $X \rightarrow A$, we must satisfy the following conditions. Suppose that $Z \rightarrow A$ is the new FD obtained after maintaining $X \rightarrow A$:

1. $Z$ should not be included in any agree set of $Ag(r)$ that does not contain $A$.
2. $Z$ should not be included in any agree generator induced by $t$ that does not contain $A$.
3. $Z \rightarrow A$ has to be minimal since it belongs to $Cover(\mathcal{F}_{r \cup t})$.

Through Proposition 3, we point out that maintaining a violated FD can be reduced to the problem of computing minimal transversals of a hypergraph.

**Proposition 3.** *Let* $X \rightarrow A$ *be a minimal FD of* $Cover(\mathcal{F}_r)$ *violated after the insertion of* $t$. $MG_1, MG_2, \ldots, MG_n$ *are the maximal agree generators induced by* $t$ *that do not contain* $A$ *and* $L_1, L_2, \ldots, L_n$ *are the antecedents of the FDs obtained when* $MG_1, MG_2, \ldots, MG_n$ *are, respectively, considered. Then,* $X \rightarrow A$ *will be replaced by the following minimal FDs:*

$\{Z \rightarrow A | Z \in L_n\}$ *such that:*
$L_0 = X$ *and* $L_i = Min\{T \cup \{e\} | T \in L_{i-1}, \{e\} \in \overline{MG_i}\}$ ($\overline{MG_i}$ *is the complementary set of* $MG_i : \mathcal{A} - MG_i$).

*Proof. As we said before, a violated FD* $X \rightarrow A$ *has to be replaced by* $Z \rightarrow A$ *where* $Z$ *is not contained in any set of* $Ag(r \cup t)/A$ (*the agree sets that do not contain* $A$). *Furthermore,* $Z$ *has to be minimal. Dually,* $Z$ *should have a non empty intersection with each complemented set of* $Ag(r \cup t)/A$ (*the complementary set of* $X$, *denoted* $\overline{X} = \mathcal{A} - X$). *Hence,* $Z$ *is a minimal transversal of the hypergraph* $\mathcal{H}$ = ($\mathcal{A}$, $\{\overline{Ag}_{-m}, \ldots, \overline{Ag}_0, \overline{G}_1 \ldots, \overline{G}_n\}$) *where* $\forall i \in [-m..0]$, $Ag_i$ *are the agree sets of* $r$ *which do not contain* $A$ *and* $\forall j \in [1..n]$, $G_j$ *are the agree generators induced by* $t$ *which do not contain* $A$. *According to Proposition 1,* $Z$ *is a minimal transversal of* $Min(H)$ =($\mathcal{A}$, $\{\overline{MAg}_{-m}, \ldots, \overline{MAg}_0, \overline{MG}_1 \ldots, \overline{MG}_n\}$) *where* $\forall i \in [-m..0]$, $MAg_i$ *are the maximal agree sets of* $r$ *which do not contain* $A$ *and* $\forall j \in [1..n]$, $MG_j$ *are the maximal agree generators induced by* $t$ *which do not contain* $A$. *According to Proposition 3,* $Tr(Min(\mathcal{H}))$ *can be determined incrementally. Indeed, let us define a sequence* $\{L_{-m}, \ldots, L_0, \ldots, L_n\}$ *are the transversal hypergraph obtained when* $\overline{MAg}_{-m}, \ldots, \overline{MAg}_0, \overline{MG}_1 \ldots, \overline{MG}_n$ *are, respectively, considered. It is obvious that* $X \in L_0$ (*since* $L_0$ *contains all minimal transversals of the complementary sets of* $Ag(r)^A$). *Hence, in order to update* $X \rightarrow A$, *we have to restrict* $L_0$ *to* $X$ *and continue to compute incrementally* $Tr(\mathcal{G})$ *where* $\mathcal{G}$ =($\mathcal{A}$, $\{\overline{MG}_1 \ldots, \overline{MG}_n\}$), $L_0 = X$ *and* $L_i = Min(T \cup \{e\} | T \in L_{i-1}, \{e\} \in \overline{MG_i})$.

**Example 10.** *Let us consider the relation* $r$ *of Table 1 and suppose that it contains only the five first tuples and we will insert the tuple* $t_6$. *The following table gives* $Cover(\mathcal{F}_r)$.

| | | | |
|---|---|---|---|
| $BC \rightarrow A$ | $AB \rightarrow C$ | $A \rightarrow D$ | $AB \rightarrow E$ |
| $BD \rightarrow A$ | $BD \rightarrow C$ | $C \rightarrow D$ | $BD \rightarrow E$ |
| $BE \rightarrow A$ | $E \rightarrow C$ | $E \rightarrow D$ | $C \rightarrow E$ |

*After computing the agree sets induced by $t_6$, we obtain $Ag(r)_{t_6} = \{C, E, BC, AD\}$. After the identification of the agree generators induced by $t_6$, we obtain $Gen(r)_{t_6} = \{C, E, BC\}$. Thanks to this set, we can deduce FDs which will be violated after the insertion of $t_6$. Indeed, $BC \rightarrow A, E \rightarrow C, C \rightarrow D, E \rightarrow D$ and $C \rightarrow E$ will be violated. Let us maintain $C \rightarrow E$. The maximal agree generators induced by $t_6$ that do not contain $E$ are $\{BC\}$. Hence, the complemented set of $BC$ is $ADE$. $C \rightarrow E$ will be replaced by the following minimal FDs: $\{Y \rightarrow E | Y \in L_1\}$ such that $L_0 = C$ and $L_1 = Min\{AC, CD, CE\}$. Consequently, $C \rightarrow E$ will be replaced by $AC \rightarrow E$ and $CD \rightarrow E$.*

Proposition 3 underpins the procedure of maintaining a violated functional dependency $X \rightarrow A$ which uses a depth-first search. Indeed we use a search tree having $X$ as root and whose leaf nodes represent candidate antecedents of FD which would replace $X \rightarrow A$. An arbitrary node, in level $i$, represents an antecedent obtained by considering maximal agree generators induced by $t$ $MG_1, MG_2, \ldots, MG_n$. Indeed, the nodes of level $i$ represent the set $L_i$ (*c.f.,* Proposition 3). In order to obtain a candidate of level $i+1$, from a node $Y$ of level $i$, we should consider the $i+1^{th}$ maximal agree generator induced by $t$. We distinguish two cases:

- If $Y$ is a minimal transversal of $\overline{MG_{i+1}}$, then this maximal agree generator induced by$t$ is ignored.
- Else, we generate a child node equal to the union of $Y$ and $\{e\}$, such that $\{e\} \in \overline{MG_{i+1}}$.

For each leaf node $Y$, we have to verify that it does not exist a FD of $Cover(\mathcal{F}_{r \cup t})$ having an antecedent included in $Y$.

**Example 11.** *Let us maintain $C \rightarrow E$ violated after the insertion of tuple $t_6$. The maximal agree generators induced by $t_6$ that do not contain $E$ are $\{BC\}$. Firstly, we initialize the search tree by the root labeled $C$. We check if $C$ is a minimal transversal of $ADE$ (the complemented of $BC$). This test fails. Consequently, we generate a child node $AC$ that represents the antecedent of the functional dependency $AC \rightarrow E$. After, we generate a second child node $CD$ that represents the antecedent of the functional dependency $CD \rightarrow E$. We do not generate the child $CE$ since $E$ belongs to $CE$.*

### *3.3. Illustrative example*

In the following, we present an example that illustrates the whole process of FDs updating. Let us consider the relation of Table 1 and suppose that it contains only the five first tuples and we will insert the tuple $t_6$. The following table gives $Cover(\mathcal{F}_r)$.

| | | | |
|---|---|---|---|
| $BC \rightarrow A$ | $AB \rightarrow C$ | $A \rightarrow D$ | $AB \rightarrow E$ |
| $BD \rightarrow A$ | $BD \rightarrow C$ | $C \rightarrow D$ | $BD \rightarrow E$ |
| $BE \rightarrow A$ | $E \rightarrow C$ | $E \rightarrow D$ | $C \rightarrow E$ |

Initially, we start by reducing the relation $r$. For that, we generate the equivalence classes of $r$ with respect to $t_6$.
$r(A)_{t_6} = \{t_5\}$;
$r(B)_{t_6} = \{t_4\}$;
$r(C)_{t_6} = \{t_1, t_2, t_4\}$;
$r(D)_{t_6} = \{t_5\}$;
$r(E)_{t_6} = \{t_3\}$.
Then, $EC(r)_{t_6} = \{r(A)_{t_6}, r(B)_{t_6}, r(C)_{t_6}, r(D)_{t_6}, r(E)_{t_6}\}$.

After, we check whether $Cover(\mathcal{F}_r)$ could be the canonical cover of $(r \cup t_6)$. In the beginning, all minimal FDs of $Cover(\mathcal{F}_r)$ are candidate to be violated by $(r \cup t_6)$. In order to reduce this search space, we apply a first refinement based on the agree sets induced by $t_6$. Tanks, to the maximal equivalence classes of $r$ with respect to $t_6$, one can deduce all tuples which share with $t_6$ the value of at least one attribute. Indeed, $MC(r)_{t_6} = \{\{t_1, t_2, t_4\}; \{t_5\}; \{t_3\}\}$. Hence, $Ag(t_1, t_6) = C$ (since $t_1$ belongs only to $r(C)_{t_6}$) ; $Ag(t_2, t_6) = C$ (since $t_2$ belongs only to $r(C)_{t_6}$); $Ag(t_4, t_6) = BC$ (since $t_4$ belongs to $r(B)_{t_6}$ and $r(C)_{t_6}$); $Ag(t_5, t_6) = AD$ (since $t_5$ belongs to $r(A)_{t_6}$ and $r(D)_{t_6}$); $Ag(t_3, t_6) = E$ (since $t_1$ belongs only to $r(E)_{t_6}$). $Ag(r)_{t_6} = \{C, BC, AD, E\}$. Hence, we are sure that all minimal FDs having a consequent included in all agree sets induced by $t_6$ are not violated by $(r \cup t_6)$. For this case, the first level of refinement does not prune any minimal FD.

After, we apply a second refinement based on the agree generators induced by $t_6$. For that, we have first to identify the agree generators induced by $t_6$. For $C, (C_{Cover(\mathcal{F}_r)})^+ = CDE$. Then, $C$ is an agree generator induced by $t_6$. For $BC, (BC_{Cover(\mathcal{F}_r)})^+ = ABC$. Then, $BC$ is an agree generator induced by $t_6$. For $AD, (AD_{Cover(\mathcal{F}_r)})^+ = AD$. Then, $AD$ is not an agree generator induced by $t_6$. For $E, (E_{Cover(\mathcal{F}_r)})^+ = CDE$. Then, $E$ is an agree generator induced by $t_6$. Hence, $Gen(r)_{t_6} = \{C, BC, E\}$. Consequently, $BD \rightarrow A$, $BE \rightarrow A$, $AB \rightarrow C$, $BD \rightarrow C$, $A \rightarrow D$, $AB \rightarrow E$, $BD \rightarrow E$ are not violated by $(r \cup t_6)$ because, for these FD, it does not exist an agree generator induced by $t_6$ that contains their antecedents and does not contain their consequents. Thus, only $BC \rightarrow A$, $E \rightarrow C$, $C \rightarrow D$, $E \rightarrow D$ and $C \rightarrow E$ have to be updated. Consequently, $Cover(\mathcal{F}_r)$ cannot be the canonical cover of $(r \cup t6)$. However, we initialize $Cover(\mathcal{F}_{r \cup t_6})$ by the non violated minimal FDs of $Cover(\mathcal{F}_r)$. Thus, $Cover(\mathcal{F}_{r \cup t_6})$ is initially as follows:

| | | | |
|---|---|---|---|
| $BE \rightarrow A$ | $AB \rightarrow C$ | $A \rightarrow D$ | $AB \rightarrow E$ |
| $BD \rightarrow A$ | $BD \rightarrow C$ | | $BD \rightarrow E$ |

After, we maintain the violated minimal FDs. For $BC \rightarrow A$, the maximal agree generators induced by $t_6$ that do not contain $A$ are $\{C, BC, E\}$. We build a tree having $BC$ as root and we check if $BC$ is a minimal transversal of $ABDE$ (the complemented set of $C$). The test succeeds. Then $ABDE$ is ignored. We check if $BC$ is a minimal transversal of $ADE$ the complemented set of $BC$). The test fails. Then, we would have three child nodes: $ABC, BCD, BCE$. The first child node is ignored since the consequent belongs to $ABC$. Then, we generate the second child node $BCD$. After, we check if $BCD$ is a minimal transversal of $ABCD$ the complemented set of $E$). The test succeeds. Then, $ABCD$ is ignored. Thus, $BCD$ is a leaf node. However, we note that $BD \rightarrow A$ belongs already to $Cover(\mathcal{F}_{r \cup t_6})$ and $BD \subset BCD$. Then, $BCD$ is ignored since $BCD \rightarrow A$ is not minimal. After, we generate the third child node $BCE$. After, we check if $BCE$ is a minimal transversal of $ABCD$. The test succeeds. Then, $ABCD$ is ignored. Thus, $BCE$ is a leaf node. However, we note that $BE \rightarrow A$ belongs already to $Cover(\mathcal{F}_{r \cup t_6})$ and $BE \subset BCE$. Then, $BCE$ is ignored since $BCE$ A is not minimal.

After updating the whole set of violated FDs, we obtain the following $Cover(\mathcal{F}_{r \cup t_6})$.

| | | | |
|---|---|---|---|
| $BE \rightarrow A$ | $AB \rightarrow C$ | $A \rightarrow D$ | $AB \rightarrow E$ |
| $BD \rightarrow A$ | $BD \rightarrow C$ | $CE \rightarrow D$ | $BD \rightarrow E$ |
| | $AE \rightarrow C$ | $BE \rightarrow D$ | $AC \rightarrow E$ |
| | $DE \rightarrow C$ | | $CD \rightarrow E$ |

## 4. Experimental evaluations

In this section, we aim to show the effectiveness of INCFDS algorithm compared to the non-incremental approach. For this purpose, we conducted a set of experiments comparing INCFDS performances to those of the non incremental algorithm FASTFDS [23]. This choice is motivated by features shared by both algorithms. We implemented both algorithms in Java on a PC equipped with a 2 GHz Intel processor and 3 GB of main memory. Firstly, we give an overview of the FASTFDS algorithm against which we compare the performances of INCFDS. Then, in order to control various parameters during the tests, we carried out experiments on synthetic databases. By this way, the pros and cons for the two algorithms can be studied in depth. Finally, we illustrate the sensitivity of both approaches to some real world data.

### *4.1. The FastFDs algorithm*

Several algorithms for discovering functional dependencies have been presented [12,15,16,21,23,24]. However, we choose to compare INCFDS to FASTFDS because they share some features. Indeed, they are agree sets based algorithms; moreover, they use a depth-first search to discover the canonical cover of the FDs. Given a relation $r$, FASTFDS algorithm starts by computing the *difference sets of* $r$ which are the complemented sets of the agree sets. After, it computes the canonical cover of FDs holding on $r$ by using a depth-first search strategy [23].

### *4.2. Synthetic databases*

Our first set of experiments involved integer-valued relations. We firstly create a table with $|\mathcal{A}|$ attributes in the database and then insert $|r|$ tuples one by one. 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. After each insertion, we record the execution time of both algorithms INCFDS and FASTFDS. The experiments were carried out on: (1) a relation composed of 10000 tuples, 20 attributes and 50% of identical values per attributes. (2) a relation composed of 10000 tuples, 20 attributes and 25% of identical values per attributes. (3) a relation composed of 10000 tuples, 10 attributes and 50% of identical values per attributes.

When we observed the results of the experiments conducted on integer-valued relations, we noted that the length of agree sets is not large (*e.g.,* for a relation of 10000 tuples, 20 attributes and 50% of identical values, the maximal length of agree sets is 3). Consequently, the number of the minimal FDs is few. Moreover, the lengths of their antecedents are not large. These two reasons explain the fact that over 90% of time is spent for computing agree sets. However, it would be interesting to study the behavior of the both algorithms when the number of the violated minimal FDs is large. For that, we used a second kind of database where over 90% of time is spent for searching the minimal FDs from agree sets: the Bernoulli relations. This kind of relations involves only two values and they are similar to the transactional databases, used in the problem of association rules mining [2]. For $|\mathcal{A}|$ attributes, the expected average length of the antecedents of minimal FDs is $|\mathcal{A}|/2$ when the number of tuples $|r| = 2^{|\mathcal{A}|/4}$. The number of minimal FDs increases exponentially with $|\mathcal{A}|$.

Figure 3 depicts the performances of INCFDS and FASTFDS. The curves (1), (2) and (3) illustrate the performances of both algorithms for, respectively, a relation composed of 10000 tuples, 20 attributes and 50% of identical values per attributes; a relation composed of 10000 tuples 20 attributes and 25% of identical values per attributes and a relation composed of 10000 tuples 10 attributes and 50% of

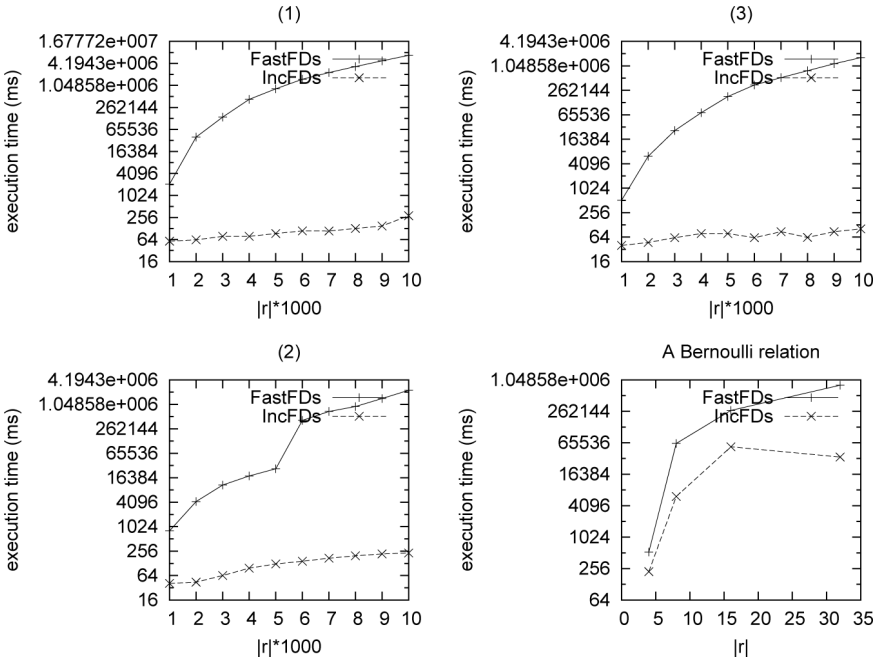


Fig. 3. The performances of FASTFDS and INCFDS.

identical values per attributes. The curve Bernoulli illustrate a Bernoulli relation having 32 tuples and 20 attributes.

Through Fig. 3, we can note that:

- the more the size of the database increases, the more the gap between FASTFDS and INCFDS increases.
- For the integer-valued relations ((1),(2) and (3)), the more the rate of identical values increases, the more the gap between FASTFDS and INCFDS increases.
- FASTFDS is more sensitive to the variation of the relation size than INCFDS.
- The execution time of FastFDs could decrease when the relation size increases.

In order to explain the behavior of both algorithms, depicted by Fig. 3, we provide additional details that allow us to study in depth the algorithms performances. These details are given by Tables 2 and 3. Table 2 reports the details when the $1000^{\text{th}}$, $4000^{\text{th}}$, $7000^{\text{th}}$ and $10000^{\text{th}}$ tuple is added to an integer-valued relation where $|\mathcal{A}| = 20$ and $c = 50\%$. Table 3 reports details when the $4^{\text{th}}$, $8^{\text{th}}$, $16^{\text{th}}$ and $32^{\text{nd}}$ tuple is added to a Bernoulli relation where $|\mathcal{A}| = 20$ and $|r| = 32$. For both tables, the second column gives the execution time for computing agree sets of the whole relation. The third column illustrates the execution time of FASTFDS. The fourth column gives the number of couples of tuples which are agree on at least one attribute of the relation. The fifth column gives the number of agree sets of the relation. The sixth column reports the average size of agree sets. The seventh column presents the size of $Cover(\mathcal{F}_r)$. The eighth column shows the execution time for computing agree sets induced by $t$ (the new tuple to add), the ninth column gives the execution time of INCFDS. The tenth column reports the number of tuples that are agree with $t$. The eleventh column reports the average size of agree sets induced by $t$. The twelfth column gives the number of agree sets induced by $t$. The thirteenth column shows the number of the violated FDs.

Through Tables 2 and 3, we can note that:

Table 2
FASTFDs versus INCFDs ($c = 50\%$ and $|\mathcal{A}| = 20$)

| | FastFDs | | | | | | IncFDs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| The inserted tuple $t$ | Execution time of agree sets computing (ms) | The whole execution time (ms) | The nbr of couples | The number of agree sets | The average size of agree sets | $\|Cover(F_r)\|$ | Execution time of agree sets induced by t (ms) | The whole execution time (ms) | The nbr of tuples agree with $t$ | The number of agree sets induced by $t$ | The average size of agree sets induced by t | The number of the violated FD's |
| 1000 | 1907 | 2101 | 1765 | 22 | 2 | 3384 | 48 | 51 | 2 | 2 | 1 | 0 |
| 4000 | 435631 | 435882 | 21885 | 50 | 2 | 2931 | 62 | 78 | 4 | 4 | 1 | 2 |
| 7000 | 2355990 | 2356303 | 55572 | 70 | 2 | 2826 | 78 | 109 | 10 | 10 | 2 | 5 |
| 10000 | 6923988 | 6923988 | 99878 | 112 | 3 | 3802 | 117 | 287 | 14 | 14 | 2 | 51 |

Table 3
FASTFDS versus INCFDS for a Bernoulli relation ($|r| = 32$ and $|\mathcal{A}| = 20$)

| | FastFDs | | | | | | IncFDs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| The inserted tuple $t$ | Execution time of agree sets computing (ms) | The whole execution time (ms) | The nbr of couples | The number of agree sets | The average size of agree sets | $\|Cover(F_r)\|$ | Execution time of agree sets induced by t (ms) | The whole execution time (ms) | The number of tuples agree with $t$ | The number of agree sets induced by $t$ | The average size of agree sets induced by t | The number of the violated FD's |
| 4 | 17 | 529 | 6 | 6 | 9 | 931 | 16 | 223 | 3 | 3 | 4 | 180 |
| 8 | 25 | 63776 | 28 | 28 | 10 | 5162 | 16 | 6127 | 15 | 15 | 5 | 1091 |
| 16 | 37 | 273481 | 120 | 120 | 12 | 27020 | 18 | 54818 | 14 | 14 | 10 | 5826 |
| 32 | 87 | 829068 | 496 | 496 | 16 | 50945 | 27 | 34871 | 31 | 31 | 6 | 6120 |

- For FASTFDS, the more the size of the relation increases the more the number of couples of tuples, the number of agree sets and thus, execution time increase. Indeed, by augmenting the relation with new tuples, we increase the chance of having tuples that are agree on at least one attribute of the relation. Hence, the number of couples of tuples would increase. Consequently, the number of agree sets and the time required for their computing would increase. For the integer-valued relation, where the average size of agree sets is not large, computing the canonical cover from agree sets is not costly. However, for Bernoulli relations, the average size of agree sets is large and the computing of the canonical cover from agree sets is costly. Hence, we can deduce that performances of FASTFDS algorithms are sensitive to the variation of the relation size, the rate of identical values and the average size of the agree sets.
- For INCFDS, the more the size the number of tuples sharing with $t$ at least the value of one attribute increases the more the number of agree sets induced by $t$ and the time required for their computing increase. Moreover, we note that the more the average size of agree sets induced by $t$ and the number of violated minimal FDs increase the more the time required for computing the new canonical cover increases. Hence, we can conclude that performances of INCFDS algorithms are sensitive to the variation number of tuples sharing with $t$ at least the value of one attribute, the average size of the agree sets induced by $t$ and the number of violated minimal FDs.

### *4.3. Real world databases*

Hereafter, we illustrate the sensitivity of FASTFDS and INCFDS to some real world data available in the repository of machine learning databases. For that, we choose the following databases: (1) `Mushrooms` that includes 8124 descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family; (2) `Nursery` developed to rank applications for nursery schools.

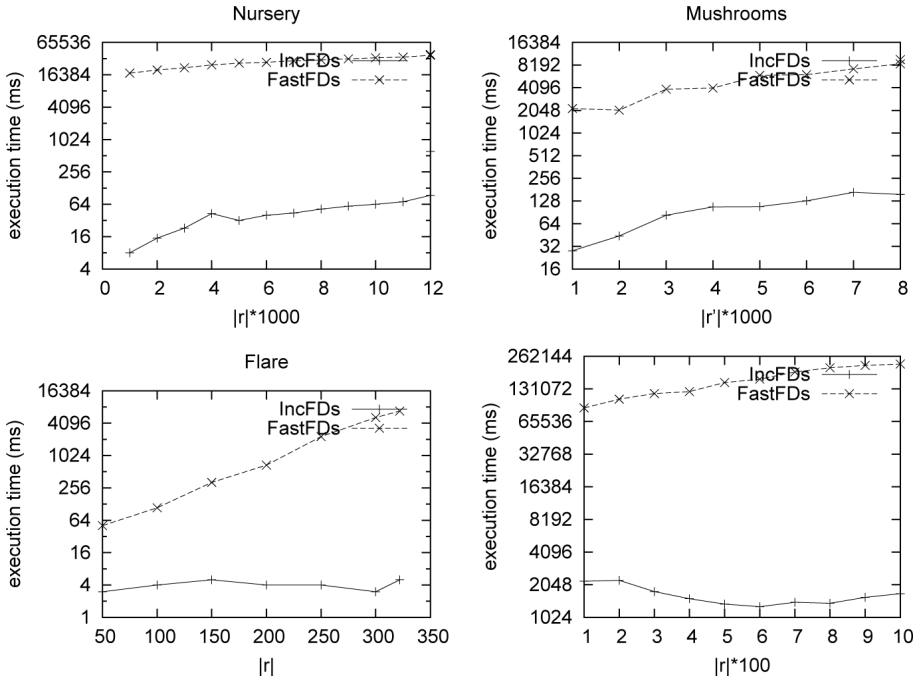


Fig. 4. The sensitivity of FASTFDS and INCFDS to some real world data.

It includes 12960 tuples and 8 attributes; (3) `Flare` that concerns solar flare. It contains 323 tuples and 10 attributes; (4) `Credit` which concerns credit card applications. It has 1000 tuples and 15 attributes.

According to curves of Fig. 4, we can note that INCFDS outperforms FASTFDS algorithm.

## 5. Conclusion and perspectives

Dynamicity is an important feature of real life database that should be taken into consideration by functional dependencies mining approaches. In this paper, we proposed the first algorithm, called INCFDS to maintain the canonical functional dependencies incrementally, when a new tuple is appended to the original database. Indeed, given a relation $r$ and a new tuple $t$, we propose to discover the canonical cover of the FDs holding on $(r \cup t)$ by taking advantages of the canonical cover of FDs holding on $r$. For that, we start by verifying that the updated relation does not violate any FD of the canonical cover of FDs holding on $r$. Whenever there are some FDs violated by the updated relation, we maintain partially the canonical cover of FDs holding on $r$ in order to deduce the canonical cover of FDs holding on $(r \cup t)$. Experimental results revealed benefits of using incremental approach to mine the canonical cover of FDs.

In this paper, we addressed a particular case of update, which consists in adding a single new tuple. However, it would be interesting to setting up a thorough framework which can handle all the update cases including insertion, deletion and modification of a set of tuples. The study of this issue is currently under investigation. However, it is important to mention that the insertion of a set of tuples can be easily addressed by changing slightly the INCFDS algorithm. Indeed, we have only to compute the agree sets induced by the whole set of inserted tuples instead of the agree sets induced by $t$. Another extension would concern the study of incremental mining of approximate dependencies. Furthermore, the study of possible parallelization of INCFDS should be investigated.

## References


[1] S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases, in: *the ACM SIGMOD Intl. Conference on Management of Data, Washington, USA*, 1993, pp. 207–216.

[3] J. Baixeries, A formal concept analysis framework to mine functional dependencies, in: *Proceeding of the Workshop on Mathematical Methods for Learning, Villa Geno, Italy*, 2004.

[4] C. Berge, *Hypergraphs: Combinatorics of Finite Sets, volume 45 of North Holland Mathematical Library*, Elsevier Science, 1989.

[5] E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in: *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '02, London, UK, Springer-Verlag, 2002, pp. 133–141.

[6] J. Demetrovics, L. Libkin and I.B. Muchnik, Functional dependencies in relational databases : A lattice point of view, *Discrete Applied Mathematics* **40** (1992), 155–185.

[7] T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, *SIAM Jounal of Computing* **24**(6) (1995), 1278–1304.

[8] T. Eiter and G. Gottlob, Hypergraph transversal computation and related problems in logic and ai, in: *Proceedings of the European Conference on Logics in Artificial Intelligence*, JELIA '02, London, UK, Springer-Verlag, 2002, pp. 549–564.

[9] S. Flesca, F. Furfaro, S. Greco and E. Zumpano, Repairing inconsistent xml data with functional dependencies, in: *Encyclopedia of Database Technologies and Applications*, L.C. Rivero, J.H. Doorn and V.E. Ferraggine, eds, Idea Group, 2005, pp. 542–547.

[10] S. Guirguis, R. Kulkarni and S. Chang, A multimedia data streams model for content-based information retrieval, in: *Proceedings of the 14th International Conference on Distributed Multimedia Systems*, DMS 2008, September 4–6, 2008, Boston, Massachusetts, USA, 2008, pp. 232–239.

[11] M. Heikki and K. Räihä, Algorithms for inferring functional dependencies from relations, *Data Knowledge Engeneering* **12**(1) (1994), 83–99.

[12] Y. Huhtala, J. Kärkkäinen, P. Porkka and H. Toivonen, Tane: An efficient algorithm for discovering functional and approximate dependencies, *Computer Journal* **42**(2) (1999), 100–111.

[13] D. Johnson, A catalog of complexity classes, in: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity* (*A*), 1990, pp. 67–161.

[14] N. Linial and M. Tarsi, Deciding hypergraph 2-colourability by h-resolution, *Theoretical Computer Science* **38** (1985), 343–347.

[15] S. Lopes, J. Petit and L. Lakhal, Efficient discovery of functional dependencies and armstrong relations, in: *Proceedings of the 7th International Conference on Extending Database Technology*, Konstanz, Germany, March 27–31, Springer, 2000, pp. 350–364.

[16] S. Lopes, J. Petit and L. Lakhal, Functional and approximate dependency mining: database and FCA points of view, *J Exp Theor Artif Intell* **14**(2–3) (2002), 93–114.

[17] D. Maier, *The Theory of Relational Databases*, Computer Science Press, 1983.

[18] H. Mannila and K. Räihä, Design by example: An application of armstrong relations, *Journal of Computer and System Sciences* **33**(2) (1986), 126–141.

[19] H. Mannila and K. Räihä, Dependency inference, in: *Proceedings of 13th International Conference on Very Large Data Bases*, September 1–4, 1987, Brighton, England, 1987, pp. 155–158.

[20] U. Nambiar and S. Kambhampati, Mining approximate functional dependencies and concept similarities to answer imprecise queries, in: *Proceedings of the Seventh International Workshop on the Web and Databases*, WebDB 2004, June 17–18, 2004, Maison de la Chimie, Paris, France, Colocated with ACM SIGMOD/PODS 2004, 2004, pp. 73–78.

[21] N. Novelli and R. Cicchetti, Fun: An efficient algorithm for mining functional and embedded dependencies, in: *Proceedings 8th International Conference on Database Theory*, J.V. den Bussche and V. Vianu, eds, London, UK, January 4–6, volume 1973 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 189–203.

[22] H. Beng Kuan Tan and Y. Zhao, Automated elicitation of functional dependencies from source codes of database transactions, *Information and Software Technology* **46**(2) (2004), 109–117.

[23] C.M. Wyss, C. Giannella and E.L. Robertson, Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances – extended abstract, in: *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, DaWaK 2001, Munich, Germany, September 5–7, Springer, 2001, pp. 101–110.

[24] H. Yao, H.J. Hamilton and C.J. Butz, Fd_mine: Discovering functional dependencies in a database using equivalences, in: *Proceedings of the 2002 IEEE International Conference on Data Mining* (*ICDM 2002*), 9–12 December, Maebashi City, Japan, 2002, pp. 729–732.