The Computer Journal Advance Access published June 9, 2014

doi:10.1093/comjnl/bxu047

A Method to Find Functional Dependencies Through Refutations and Duality of Hypergraphs

JOEL FUENTES[1,*], PABLO SÁEZ[1], GILBERTO GUTIÉRREZ[1] AND ISAAC D. SCHERSON[2]

[1]*Department of Computer Science and Information Technologies, Universidad del Bío-Bío, Chillán, Chile*
[2]*Department of Computer Science, University of California, Irvine, CA, USA*
[*]*Corresponding author: jfuentes@ubiobio.cl*

One of the most important steps in obtaining a relational model from legacy systems is the extraction of functional dependencies (FDs) through data mining techniques. Several methods have been proposed for this purpose and most use direct search methods that traverse the search space in exponential time in the number of attributes of the relation. As it is not uncommon to find in practice relations with tens of attributes, a need exists to further develop more efficient techniques to find FDs. The method studied here finds the minimal set of minimal FDs using algorithms that solve the hypergraph duality problem applied on the complement of the refutation hypergraph of the relation without going through the exponential search space. After showing that the extraction of FDs can be reduced to the hypergraph duality problem, experimental results are given as verification and characterization of the correctness and time complexity of the proposed tool.

Keywords: functional dependencies; duality of hypergraphs; minimal transversals

Received 4 December 2013; revised 1 May 2014
Handling editor: Rada Chirkova

1. INTRODUCTION

The extraction of functional dependencies (FDs from now on) from an instance of a relation is an important data mining technique, used in database design, query optimization and reverse engineering among others [1]. Studies like [2] propose methods and strategies to obtain the relational database model from legacy systems, where one of the steps is the automatic extraction of the FDs. This shows the importance of having efficient tools that perform this task.

A number of tools and algorithms have been indeed proposed for this purpose, but most of them are exponential in time in the number of attributes of the relation. But in real situations it is common to have relations with a high number of attributes (for instance more than 20 or 30 attributes). This motivates the search for more efficient techniques than those used by these tools. We will show in the following sections that the problem of finding FDs can be efficiently reduced, by making use of FD refutations, to the well-known hypergraph duality problem [3], for which for instance $O(n^{\log n})$, i.e. quasi-polynomial algorithms are known [4]. The idea of using hypergraph transversals for inferring FDs was independently proposed in [5, 6] and *refutations* were referred to as *antikeys*. Our main contributions in this paper can be summarized as follows:

1. Implementation of an efficient computational method to obtain the set of refutations for FDs, given an instance r of a relation R.
2. A method to store and process these refutations, represented as hyperedges of a hypergraph.
3. The obtention of all the minimal FDs that are valid in an instance of a relation, by means of the computation of the set of minimal transversals of this hypergraph.
4. A complete tool to compute the set of minimal FDs, together with an analysis of the time spent by the tool on the two main processes.

The present article is divided into six sections. In Section 2, the problem is stated and related work is reviewed. In Section 3, we briefly recall the hypergraph duality problem and the known algorithms that solve it. In Section 4, we present the proposed approach to the problem. And in Sections 5 and 6 some experimental results and conclusions are given, respectively.

2. PROBLEM STATEMENT AND RELATED WORK

Let R be a relation with a set A of attributes and let r be an instance of R. An FD is an expression of the form $X \rightarrow Y$, where $X \subseteq A$ and $Y \in A$. X is called the *determinant set* and Y is called the *dependent attribute*. The dependency is *valid* in the instance r if and only if for every pair of rows (tuples) $t, u \in r$, whenever $t[B] = u[B]$ for all $B \in X$, it is also the case where $t[Y] = u[Y]$. An FD $X \rightarrow Y$ is called *trivial* if $Y \in X$. Our problem can be formally stated as follows: *given a relation R and an instance r of R, find all non-trivial FDs that are valid in r*. The following proposition will help us give a more practical formulation of the problem.

PROPOSITION 2.1. *Given two FDs $X \rightarrow B$ and $Y \rightarrow B$, if $X \subseteq Y$, then $Y \rightarrow B$ is a logical consequence of $X \rightarrow B$.*

Proof. In fact, assuming $X = \{B_1, B_2, \ldots, B_m\}$ and $Y = \{B_1, B_2, \ldots, B_n\}$, with $m \leq n$, given a relation instance r, the FDs $X \rightarrow B$ and $Y \rightarrow B$, as logical formulae, can be written, respectively, as $(\forall t, u \in r)t[B_1] = u[B_1] \wedge t[B_2] = u[B_2] \wedge \cdots \wedge t[B_m] = u[B_m] \implies t[B] = u[B]$ and $(\forall t, u \in r)t[B_1] = u[B_1] \wedge t[B_2] = u[B_2] \wedge \cdots \wedge t[B_n] = u[B_n] \implies t[B] = u[B]$, so the latter follows from the former (it has a stronger antecedent in the implication). □

So, given two FDs $X \rightarrow B$ and $Y \rightarrow B$ as in the proposition, the latter can be considered redundant, in the sense that if the former has been found, the latter does no longer need to be considered, since it follows from the former. This consideration leads us to give the following definition.

DEFINITION 2.1. *Given a relation instance r, an FD $X \rightarrow B$ that is valid in r is said to be minimal if no other FD $Y \rightarrow B$ is valid in r with Y a proper subset of X.*

So the practical formulation of our problem is as follows: *given a relation R and an instance r of R, find all non-trivial, minimal FDs that are valid in r*.

The algorithms that have been proposed to solve this problem can be classified into three categories, or approaches [7]: the first one is the generation-and-test of candidate FDs, where the TANE [8] and FUN [9] algorithms can be highlighted; the second approach is the Minimal Cover, with the algorithms proposed in by Flach and Savnik [10], Lopes *et al.* [11] and Wyss *et al.* [12]; and finally the Formal Concept Analysis, with the methods studied in [13–15]. For example, the TANE algorithm is based on an FD search within a boolean lattice, level by level. Approximated FDs can be obtained, according to an error factor, and pruning techniques are used in order to optimize the search. The problem is that the performance of the algorithm decreases when the number of attributes in a relation is big (for instance, more than 25 or 30 attributes), as shown by the experiments in [8, 16]. The reason is that the size of the boolean lattice grows exponentially with the number of attributes.

The other above-mentioned methods have similar problems. That is, they perform in a way or another a search over the boolean lattice.

2.1. The TANE algorithm

The TANE algorithm was proposed by Huhtala *et al.* [8]. It corresponds to an algorithm in the category generate-and-test, and obtains the set of minimal FDs that are valid in an instance of a relation. It performs a partition-based search of FDs. Partitions are generated from equivalence classes of tuples, according to the values that attributes can take in a tuple.

In this algorithm's proposal, the *equivalence class* of a tuple $t \in r$ with respect to the set $X \subseteq A$, denoted by $[t]_X$, is defined as $[t]_X = \{u \in r \mid t[B] = u[B], \forall B \in X\}$. Therefore, the set $\pi_X = \{[t]_X \mid t \in r\}$ of equivalence classes is a *partition* of r with respect to X. That is, π_X is a collection of disjoint sets (equivalence classes) of tuples, where the tuples in each equivalence class have the same values for the attributes in X, and the union of all the equivalence classes gives the whole relation instance r. Finally, the ranking $|\pi|$ of a partition π is the number of equivalence classes in π.

Example 2.1. In the relation of Fig. 1, attribute A has value 1 only in tuples 1 and 2, so these form an equivalence class $[1]_A = [2]_A = \{1, 2\}$, and the partition with respect to attribute A is $\pi_A = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$. Similarly, the partition with respect to B, C is $\pi_{\{B,C\}} = \{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$.

Given the above definitions, is easy to see that FD $X \rightarrow A$ is valid if and only if $|\pi_X| = |\pi_{X \cup \{A\}}|$.

With respect to the algorithm itself, its main characteristics are as follows:

1. It performs a search level by level, as shown in Fig. 2, where the computation at each level is based on the computation at the previous levels.

Tuple ID	A	B	C	D
1	1	a	top	Flower
2	1	A		Tulip
3	2	A	top	Daffodil
4	2	A	top	Flower
5	2	b		Lily
6	3	b	top	Orchid
7	3	C		Flower
8	3	C	hill	Rose

FIGURE 1. Example of a relation and its partitions.

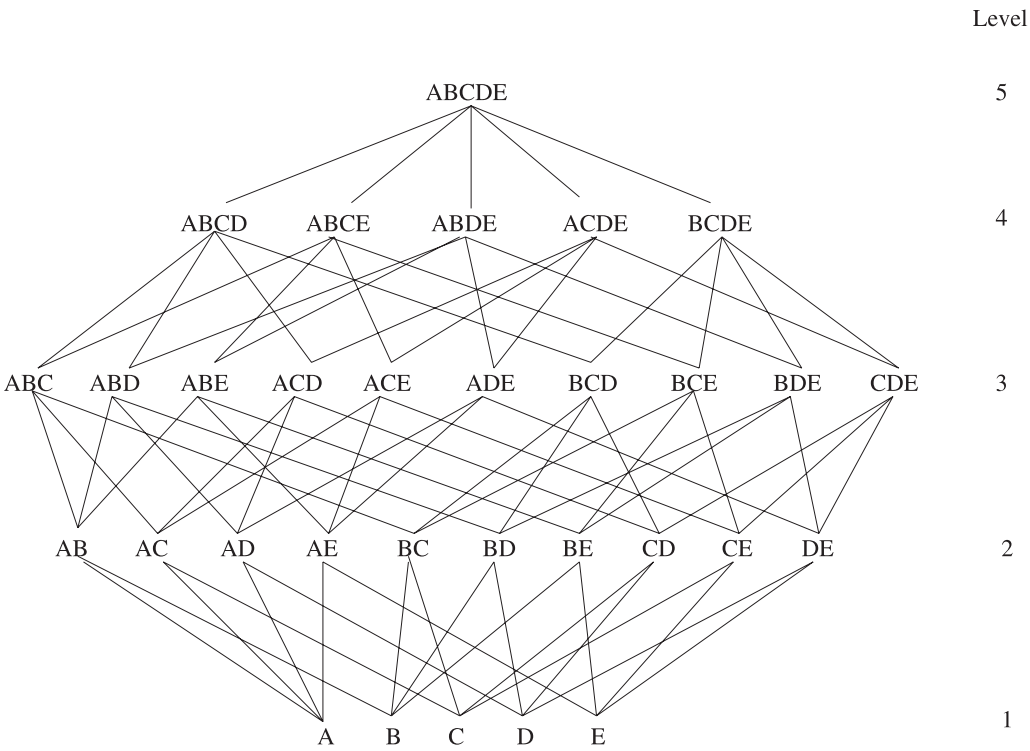

FIGURE 2. Possible combinations of the attributes.

2. It has two methods to reduce the complexity in time and space:

   (a) The first is to replace the partitions by a more compact representation, by translating the attributes into numeric values, using hash tables.
   (b) The second is to introduce a slight error e to find approximate FDs.

3. Utilization of pruning techniques to optimize the search. Basically supersets of determinant sets that are already found are discarded. That is, if $X \subseteq Z$ and $X \rightarrow Y$ is a valid FD, then $Z \rightarrow Y$ is not a minimal FD, so it can be discarded as a candidate.

To find all non-trivial minimal dependencies, TANE performs a level-by-level search. A level L_l is the collection of attribute sets of size l that can potentially be used to build dependencies based on the above considerations. TANE starts with $L_1 = \{\{A\}|A \in R\}$, and computes L_2 from L_1, L_3 from L_2, and so on. Algorithm 1 shows this procedure.

The procedure COMPUTE_DEPENDENCIES(L_l) finds the minimal dependencies in level L_l from those in L_{l-1}. The procedure PRUNE(L_l) reduces the search space and the procedure GENERATE_NEXT_LEVEL(L_l) forms the next level from the current one.

Algorithm 1. TANE.

1. $L_0 := \{\emptyset\}$
2. $C^+(\emptyset) := R$
3. $L_1 := \{\{A\}|A \in R\}$
4. $l := 1$
5. **while** $l \neq \emptyset$ **do**
6. COMPUTE_DEPENDENCIES(L_l)
7. PRUNE(L_l)
8. L_{l+1}:= GENERATE_NEXT_LEVEL(L_l)
9. $l := l + 1$
10. **end while**

3. HYPERGRAPH DUALITY

3.1. Hypergraphs

A *hypergraph* is defined as a generalized graph $H = (A, E)$, where A is a finite set of vertexes and $E \subseteq \mathcal{P}(A)$ is a set of *hyperedges* ($\mathcal{P}(C)$ is the set of subsets of C). This concept was proposed by Claude Berge in 1970 [17] and it can be considered as a generalization of the concept of graph, in the sense that the restriction that edges must always consist of two vertexes is dropped.

Example 3.1. Let $A = \{a, b, c, d, e, f\}$, and let H be the set of hyperedges $\{\{a, b\}, \{b, c\}, \{c, d, e\}, \{f\}\}$. Figure 3 represents hypergraph H by means of a Venn diagram.

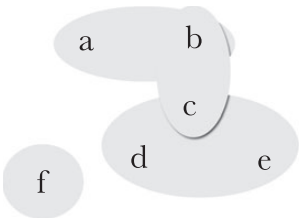

FIGURE 3. Hypergraph $H = \{\{a, b\}, \{b, c\}, \{c, d, e\}, \{f\}\}$.

We use the boolean matrix representation for hypergraphs, where each row represents a hyperedge and each column represents a vertex. For example, the following matrix represents the hypergraph from Fig. 3:

$$H = \begin{array}{c} \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}.$$

3.2. Operators on hypergraphs

Several operators have been defined for hypergraph processing. We will briefly review the relevant ones, using essentially the notation from [18]. Algorithms exist that compute these operators, although these algorithms are not polynomial in time in every case. The main operators that are relevant for the purpose of the present work are the following.

3.2.1. *The cartesian product:* $\vee$

Given two hypergraphs, $H = \{E_1, E_2, \ldots, E_m\}$ and $H' = \{F_1, F_2, \ldots, F_{m'}\}$ the cartesian product is defined as follows:

$$H \vee H' = \{E_i \cup F_j | 1 \leq i \leq m, 1 \leq j \leq m'\}. \quad (1)$$

It produces the union of all possible pairs of hyperedges, where the first hyperedge comes from the first hypergraph and the second one from the second hypergraph.

3.2.2. *The minimization of* $H : \mu(H)$

The operator μ, applied to a hypergraph H, produces a hypergraph with all minimal hyperedges from H. It is defined as

$$\mu(H) = \{X \in E; \neg\exists Y \in E, Y \subsetneq X\}. \quad (2)$$

Example 3.2. Let $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, represented as the following boolean matrix:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Then

$$\mu(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Note that $\mu(H) \subseteq H$, and that $A \in \mu(H)$ if and only if $H = \{\{A\}\}$.

The computation of $\mu(H)$ is clearly polynomial in time, since it can be done by comparing every hyperedge in H to every other hyperedge, thereby obtaining the minimal hyperedges.

3.2.3. *The clutter of* $H : \nu(H)$

The operator ν, applied to a hypergraph H, generates the set of all hyperedges that are a superset of some hyperedge in H; that is

$$\nu(H) = \{X | X \subseteq A; \exists Y \in E, Y \subseteqq X\}. \quad (3)$$

Example 3.3. Let $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, represented as a matrix as follows:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Then

$$\nu(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}.$$

Note that $H \subseteq \nu(H) \subseteq P(A)$, and that, for all H, $A \in \nu(H)$.

The computation of $\nu(H)$ is not polynomial in general, since in the worst case the cardinality of $\nu(H)$ itself can be exponential in the size of H.

3.2.4. *The inverse clutter of* $H : \nu'(H)$

The operator ν', applied to a hypergraph H, generates the set of all hyperedges that are a subset of some hyperedge in H. So, it is somehow the opposite of ν. Namely,

$$\nu'(H) = \{X | X \subseteq A; \exists Y \in E, Y \supset X\}. \quad (4)$$

Example 3.4. Let $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, represented as a matrix as follows:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Then

$$\nu'(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}.$$

3.2.5. The blocker of H : $\tau(H)$

A *transversal* of a hypergraph H is a hyperedge that intersects all hyperedges in H. The operator τ, when applied to a hypergraph H, generates precisely the set of transversals of H; that is

$$\tau(H) = \{X \subseteq A; \forall Y \in E, X \cap Y \neq \emptyset\}. \quad (5)$$

Example 3.5. Let $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, represented as a matrix as follows:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Then

$$\tau(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}.$$

The computation of $\tau(H)$, also known as the *hitting set Problem*, is not polynomial in general, since the size of $\tau(H)$ is not polynomial in the size of H in the worst case.

3.2.6. The slices of H : $\lambda(H)$

The operator λ, applied to a hypergraph H, generates the set of *minimal transversals* of H. It is defined as follows:

$$\lambda(H) = \mu(\tau(H)). \quad (6)$$

Example 3.6. Let $H' = \{\{c\}, \{b, c\}, \{a, b\}\}$, represented as a matrix as follows:

$$H' = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Then

$$\lambda(H') = \begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Note that $\lambda(H) \subseteq \tau(H) \subseteq P(A)$.

3.3. The hypergraph duality problem

The hypergraph duality problem [3] has a wealth of applications in logic, game theory, database indexation, model-based diagnosis, query optimization, artificial intelligence, machine learning, among others. See [19] for a survey of the applications of the problem.

For the purpose of our work, we will consider the following as the hypergraph duality problem: given a hypergraph H, compute $\lambda(H)$. For this problem Fredman and Khachiyan proposed in 1996 a sub-exponential algorithm [4], namely an $O(n^{\log n})$ one, which allows to conjecture that this problem is not NP-complete. But to date it remains an open question whether this problem is in P, and even whether it is in NP [3].

In recent years new algorithms have been proposed that turn out to be efficient in practice, although not necessarily having the same worst case time complexity as Fredman and Khachiyan's algorithm, such as the ones by Kavvadias and Stavropoulos [20] and by Murakami and Uno [21]. These algorithms are implemented in the tool that we propose, and will be analyzed in the following subsection.

3.4. Algorithms for the computation of minimal transversals

We start with the most elementary algorithm, which has been the basis for many subsequent ones.

3.4.1. Berge's algorithm

It was proposed by Berge [17]. We suppose given a hypergraph $H = \{E_1, \ldots, E_m\}$ over a set of vertexes A. Let $H_i = \{E_1, \ldots, E_i\}$. The algorithm iteratively computes $K_i = \lambda(H_i)$ for $i = 1, \ldots, m$, so that $K_m = \lambda(H)$. Starting from $K_1 = \{\{v\}, v \in E_1\}$, it can be seen that

$$K_i = \mu(\tau(H_{i-1}) \vee \tau(E_i)),$$
$$K_i = \mu(K_{i-1} \vee \{\{v\}, v \in E_i\}).$$

Example 3.7. Let $H = \{\{4, 5\}, \{1, 2, 3\}, \{1, 2, 5\}\}$. The application of Berge's algorithm is as follows:

1. In the first iteration we have $\{\{4\}, \{5\}\} \vee \{\{1\}, \{2\}, \{3\}\}$, which produces the set of transversals {{1,4}, {2,4}, {3,4}, {1,5}, {2,5}, {3,5}, {1,2,4}, {1,3,4}, {2,3,4}, {1,2,5}, {1,3,5}, {2,3,5}}. The application of the μ operator to this set yields the following set of minimal transversals $\{\{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 5\}, \{2, 5\}, \{3, 5\}\}$.
2. In the next iteration, we compute the cartesian product of this hypergraph with the set of minimal transversals of the last hyperedge: $\{\{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 5\}, \{2, 5\}, \{3, 5\}\} \vee \{\{1\}, \{2\}, \{5\}\}$, getting the following set of hyperedges: {{1,4}, {2,4}, {1,5}, {2,5}, {3,5}, {1,2,4}, {1,4,5}, {2,4,5}, {1,3,4}, {2,3,4}, {3,4,5}, {1,2,5}, {1,3,5}, {2,3,5}}. Finally, applying the μ operator to this set, we obtain $\lambda(H) =$

$\{\{1,4\},\{2,4\},\{1,5\},\{2,5\},\{3,5\},\{1,3,4\},\{2,3,4\},\{3,4,5\}\}$.

The main disadvantage of this algorithm is that it produces too many intermediate transversals in general. It has an exponential complexity in the worst case.

3.4.2. Algorithm by Fredman and Khachiyan

It was first described in [4] and further developed in [22]. Although we do not implement it, because it does not show a good performance in practice, we briefly mention it here, since it is the most efficient known algorithm from the perspective of the worst case time complexity. It is $O(n^{\log n})$, that is, sub-exponential but super-polynomial. The algorithm takes as input a pair of hypergraphs (H, K) and decides whether $K = \lambda(H)$. Its basic idea is the construction of a binary tree, where each node represents a Shannon decomposition of a pair of hypergraphs. The root of the tree is the original pair (H, K) and the decomposition vertex $a \in A$ is chosen at each node so as to minimize the size of the resulting decomposed pairs. The process is iterated until the size of the pairs makes the decision procedure trivial.

3.4.3. Algorithm by Kavvadias and Stavropoulos

Kavvadias and Stavropoulos' algorithm was proposed in 2005 in [20]. It is an improvement on Berge's algorithm based on the generation of generalized nodes and the use of depth-first search. We review it here since it is part of our implementation.

DEFINITION. *Given a hypergraph H over a set of vertexes A, the set $X \subseteq A$ is a Generalized Node of H if all the elements in X belong to the same hyperedges in H. That is, A is partitioned into a set of generalized nodes H_i satisfying the following properties*:

1. $A = X_1 \cup X_2 \cup \cdots X_k$;
2. $\forall i, j = 1, \ldots, k, X_i \cap X_j = \oslash$;
3. $\forall i = 1, \ldots, k, \forall a, b \in A, a \in X_i \wedge b \in X_i \Rightarrow \forall Y \in H, a \in Y \Leftrightarrow b \in Y$.

The algorithm can be described as follows. We consider a hypergraph $H = \{E_1, \ldots, E_m\}$ defined over a set A of n nodes. For each $i = 1, \ldots, m$ the algorithm computes the set of generalized nodes $\{X_1, X_2, \ldots, X_{k_i}\}$ for the partial hypergraph $H_i = \{E_1, \ldots, E_i\}$ and then $\lambda(H_i)$ is computed, where at each step the hyperedge E_{i+1} is added to H_i, since $H_{i+1} = H \cup \{E_{i+1}\}$. Adding this hyperedge can modify the set of generalized nodes. There are three possible cases for each generalized node X in H_i:

(i) (α) $X \cap E_{i+1} = \emptyset$. In this case X is also a generalized node of H_{i+1}.

(ii) (β) $X \subset E_{i+1}$. In this case, X is also a generalized node of H_{i+1}.

(iii) (γ) $X \cap E_{i+1} \neq \emptyset$ and $X \not\subseteq E_{i+1}$. In this case X is divided into $X_1 = X \setminus (X \cap E_{i+1})$ and $X_2 = X \cap E_{i+1}$. Both X_1 and X_2 are generalized nodes of H_{i+1}.

The algorithm now determines whether (α) or (β) holds for all generalized nodes in H_{i+1}, in which case all minimal transversals and E_{i+1} remain as they are. Otherwise, if (γ) holds for some generalized node X, this node is split into X_1 and X_2. By iterating this procedure, the partial hypergraphs $\lambda(H_i)$ are computed, but using generalized nodes instead of the usual nodes. This use of generalized nodes in intermediate steps considerably reduces the amount of intermediate transversals that need to be computed with respect to, for instance, the amount needed by Berge's algorithm.

Example 3.8. Assume that the two first hyperedges in a hypergraph have 100 nodes each: $E_1 = \{v_1, \ldots, v_{100}\}$ and $E_2 = \{v_{51}, \ldots, v_{150}\}$. The partial hypergraph $\{E_1, E_2\}$ has 2550 minimal transversals: 2500 with two nodes and 50 with one node. With Kavvadias and Stavropoulos' approach only two transversals need to be considered in this case, namely the sets $\{\{v_{51}, \ldots, v_{100}\}\}$ and $\{\{v_1, \ldots, v_{50}\}, \{v_{101}, \ldots, v_{150}\}\}$.

Example 3.9. Consider the hypergraph with five nodes and four hyperedges $H = \{\{1,2,3\},\{3,4,5\},\{1,5\},\{2,5\}\}$. The tree of minimal transversals that corresponds to inputting the hyperedges in this order is shown in Fig. 4. Generalized nodes are shown in circles. So, in the tree that the algorithm computes, it can be observed that, from the first hyperedge $\{1,2,3\}$, and the second one, $\{3,4,5\}$, the minimal generalized transversals $\{\{1,2\},\{4,5\}\}$ and $\{\{3\}\}$ are obtained, and so on, until $\lambda(H) = \{\{1,5\},\{1,2,4\},\{2,5\},\{1,2,3\},\{3,5\}\}$ is reached.

3.4.4. Algorithms by Murakami and Uno

Two algorithms were proposed by Murakami and Uno [21] that focus on the efficient computation of hypergraph transversals for the case of large-scale input data with a large number of output minimal transversals, introducing new search methods, new pruning methods and techniques for the minimality check.

These authors introduce the concept of *Critical Hyperedge* to allow a minimality check as follows. For a vertex subset $S \subseteq A$, $uncov(S)$ denotes the set of hyperedges that do not intersect with S, i.e. $uncov(S) = \{X | X \in H, X \cap S = \emptyset\}$. S is a transversal if and only if $uncov(S) = \emptyset$. For a vertex $v \in S$, a hyperedge $X \in H$ is said to be critical for v if $S \cap X = \{v\}$. Murakami and Uno denote the set of all critical hyperedges for v by $crit(v, S)$, i.e. $crit(v, S) = \{X | X \in H, S \cap X = \{v\}\}$. Therefore, we have the following property.

Property. S is a minimal transversal if and only if $uncov(S) = \emptyset$, and $crit(v, S) \neq \emptyset$ holds for any $v \in S$.

Example 3.10. Consider $H = \{\{1,2\},\{1,3\},\{2,3,4\}\}$, and let the transversal S be $\{1,3,4\}$. We see that $crit(1, S) = \{\{1,2\}\}$, $crit(3, S) = \emptyset$, $crit(4, S) = \emptyset$, thus S is not minimal, and we can remove either 3 or 4. For $S' = \{1,3\}$, $crit(1, S') =$

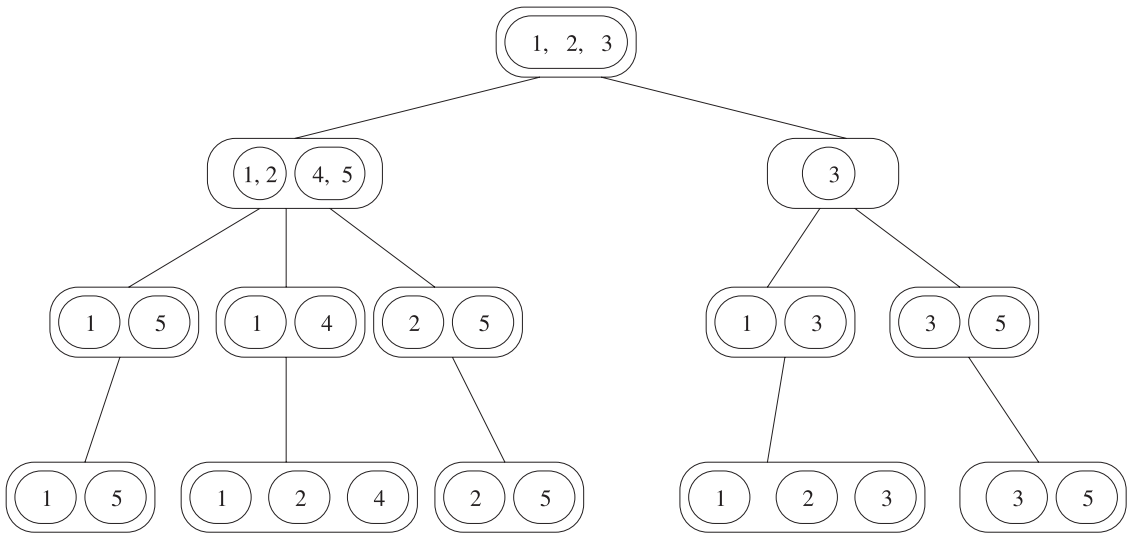

FIGURE 4. Tree of minimal transversals of $H = \{\{1, 2, 3\}, \{3, 4, 5\}, \{1, 5\}, \{2, 5\}\}$.

Algorithm 2. Update_crit_uncov(v, $crit[]$, $uncov$).

1. **for each** $X \in H(v)$ **do**
2. **if** $X \in crit[v]$ for a vertex $u \in S$ **then**
3. remove X from $crit[u]$
4. **if** $X \in uncov$ **then**
5. $uncov := uncov \setminus X$
6. $crit[e] := crit[e] \cup \{X\}$
7. **end for**

$\{\{1, 2\}\}$, $crit(3, S') = \{\{2, 3, 4\}\}$, therefore S' is a minimal transversal.

The algorithms keep lists $crit[u]$ and $uncov$ representing $crit(u, S)$ and $uncov(S)$. When the algorithm adds a vertex v to S and generates a recursive call, it updates $crit[]$ and $uncov$ by means of Algorithm 2,

Using the above minimality check, Murakami and Uno develop a depth-first search algorithm. The strategy is simple. It starts by setting S to an empty set, and adding a vertex to S one by one, maintaining the condition that any vertex in S has at least one critical hyperedge. To avoid duplications, in each iteration a list of vertices $CAND$ is used that represents the vertices that can be added in the iteration. The vertices not included in $CAND$ are not added, even if the addition satisfies the minimality condition. The algorithm can be stated as follows.

Example 3.11. Consider the hypergraph $H = \{\{1, 2\}, \{1, 3\}, \{2, 3, 4\}\}$ and an execution with $S = \{1\}$, $CAND = \{2, 3, 4\}$ and $uncov(S) = \{\{2, 3, 4\}\}$. The method $DFS(S)$ takes the next vertex from $CAND$ $v = \{2\}$ and adds it to S',

Algorithm 3. DFS(S).

global variable: $crit[]$, $uncov$, $CAND$
1. **if** $uncov = \emptyset$ **then**
2. output S; return
3. choose a hyperedge X from $uncov$
4. $C := CAND \cap F$
5. $CAND := CAND \setminus C$
6. **for each** $v \in C$ **do**
7. Update_crit_uncov(v, $crit[]$, $uncov$);
8. **if** $crit(f, S') \neq \emptyset$ for each $f \in S$ **then**
9. $DFS(S \cup v)$
10. $CAND := CAND \cup v$
11. recover the change to $crit[]$ and $uncov$ done in 7
12. **end for**

checking its minimality. Then, in the next call to $DFS(S \cup v)$ the set $S = \{1, 2\}$ is a minimal transversal because $uncov(S) = \emptyset$, and $crit(v, S) \neq \emptyset$ holds for any $v \in S$.

The second algorithm is based on a reverse search and it can be regarded as an improved version of the KS algorithm. Let $\mathcal{S} = \bigcup_{i=1}^{m} H_i$ be the set of vertex subsets that are considered by Kavvadias and Stavropoulos' algorithm. Murakami and Uno denote the minimum i, such that $H_i \in crit(v, S)$, by $min\_crit(v, S)$, and the minimum i, such that $H_i \in uncov(S)$, by $min\_uncov(S)$.

Suppose that $min\_crit(v, S) < min\_uncov(S)$ holds for any $v \in S$. Then, we can see that $crit(v, S) \neq \emptyset$ for any $v \in S$ because $min\_crit(v, S) < m + 1$. Let $i = min\_uncov(S) - 1$. Note that $i \leq m$. We can then see that S is a transversal of H_i and $min\_crit(v, S) \leq i$. This in turn implies that S is a minimal transversal in H_i and, thus, it belongs to $\mathcal{S}$. The depth-first search

starts from the emptyset. When it visits a vertex subset S, it finds all children of S iteratively and generates a recursive call for each child. The algorithm is as follows.

Algorithm 4. RS(S).

global variable: $crit[]$, $uncov$
1. **if** $uncov = \emptyset$ **then**
2. output S; return
3. $i := min\{j | H_j \in uncov\}$
4. **for each** $v \in H_i$ **do**
5. Update_crit_uncov(v, $crit[]$, $uncov$)
6. **if** $min\{t | H_t \in crit[f]\} < i$ for each $f \in S$ **then**
7. $RS(S \cup v)$
8. recover the change to $crit[]$ and $uncov$ done in 5
9. **end for**

Example 3.12. Consider the hypergraph $H = \{\{1, 2\}, \{1, 3\}, \{2, 3, 4\}\}$ and an execution initially with $S = \emptyset, crit = \emptyset$ and $uncov(S) = \{\{1, 2\}, \{1, 3\}, \{2, 3, 4\}\}$. The method $RS(S)$ takes the minimum hyperedge from $uncov(S)$ and its first vertex ($S = \{1\}$). It updates the sets $uncov$ and $crit[]$, so that they satisfy the condition of the step 6, $min\_crit(v, S) < min\_uncov(S)$, and it produces a recursive call for $RS(S')$. The next execution considers $uncov = \{\{2, 3, 4\}\}$ and the method takes this hyperedge from $uncov$ and its first vertex, getting $S = \{1, 2\}$ and the satisfaction of the condition in step 6, $min\_crit(v, S) < min\_uncov(S)$ (considering that if $uncov(S)$ is empty, $min\_uncov(S)$ is defined as $m + 1$). Finally, the algorithm outputs the minimal transversal $\{1, 2\}$ and continues with the next vertex for the other minimal transversals.

Both algorithms perform pruning methods to avoid unnecessary recursive calls. They introduce a lexicographic depth-first search, removing vertices that can never be used and prune branches without necessary vertices. The pruning drastically reduces the computation time.

4. METHOD TO FIND FDS

Many problems have an efficient solution based on direct and simple methods. In other cases, such as the one we address in this paper, indirect methods can prove more efficient. We use the concept of *refutations* (also known as *antikeys*) of FDs, assuming that an FD holds in an instance of a relation if and only if there is no refutation for this FD in the instance. The soundness of step 6 of our method can be seen as a consequence of Eiter and Gottlob [6], Theorem 7.3, where a reduction of our problem to the hypergraph duality problem is given. Direct methods, such as the ones reviewed in Section 2, are based on generating all possible combinations of attributes, which are in general exponential in number. The process is more efficient if it starts from refutations of FDs.

4.1. Description of the method

Given a set of attributes A and an instance r of a relation R that has A as its set of attributes, a *refutation* $X \nrightarrow Y$, where $X \subseteq A$ and $Y \in A$ holds if and only if there exist two tuples t and u in r such that $(\forall B \in X)\ t[B] = u[B] \wedge t[Y] \neq u[Y]$.

We describe in what follows the method that we use to extract FDs from an instance r of a relation based on refutations of FDs and hypergraph duality. We assume that an attribute $D \in A$ is given, and we compute all FDs that hold in r that are of the form $X \rightarrow D$, i.e. that have D as its dependent attribute.

1. We consider the relation instance r where the attributes have been previously codified, so that we do not work with real values but with codes that use less memory space (for instance, numbers instead of strings).

We make use of the following (simple) lemma.

LEMMA 4.1. *The FDs are preserved if the attributes are coded as integers, in a one-to-one relation.*

Proof. In fact, given a one-to-one coding, the resulting table is isomorphic to the initial one, so all of its logical properties are preserved, in particular the FDs. □

That is to say, this step of the method is sound.

2. We obtain the set of refutations of FDs of the form $X \nrightarrow D$, with $X \subset A$, that is, the set of pairs of tuples $(t, u) \in r \times r$ such that $(\forall B \in X) t[B] = u[B] \wedge t[D] \neq u[D]$.

   Observe that this process is quadratic in the number of tuples in r.
3. From this set, we consider only the *maximal* refutations, i.e. we delete all refutations that are a subset of another refutation (for the same dependent attribute D). For example, if we have the refutations $A, B, C \nrightarrow D$ and $A, B \nrightarrow D$, we need only consider $A, B, C \nrightarrow D$. We obtain a *hypergraph* over the set of nodes $A \backslash \{D\}$ where each hyperedge is the determinant set of a maximal refuted FD. Note that if $H = \{ X_1, X_2, \ldots, X_k \}$ is this hypergraph with maximal refutations, then all the FDs of the form $X \rightarrow D$, where X is a subset of some $X_i \in H$, are also refuted. For example, if $H = \{\{A, B, C\}\}$, that is, if we have $\{A, B, C\} \nrightarrow D$, then we also have $\{A, B\} \nrightarrow D$, $\{B, C\} \nrightarrow D$, and so on.

The soundness of this step of the method is given by the following lemma.

LEMMA 4.2. *Given two refutations $X \nrightarrow D$ and $Y \nrightarrow D$, if $Y \subseteq X$, then $Y \nrightarrow D$ follows from $X \nrightarrow D$.*

Proof. If $X = \{B_1, B_2, \ldots, B_m\}$ and $Y = \{B_1, B_2, \ldots, B_n\}$, with $n \leq m$, then $X \not\rightarrow D$, as a logical formula, is $(\exists t, u \in r)t[B_1] = u[B_1] \wedge \cdots \wedge t[B_m] = u[B_m] \wedge t[D] \neq u[D]$, and $Y \not\rightarrow D$, as a logical formula, is $(\exists t', u' \in r)t'[B_1] = u'[B_1] \wedge \cdots \wedge t'[B_n] = u'[B_n] \wedge t'[D] \neq u'[D]$; so the latter follows from the former. □

4. We generate hypergraph H as a boolean matrix, as defined in Section 3.1. We go through all pairs $(t, u) \in r \times r$, and, for each produced refuted FD X, we need to compare X to all X_i so far generated in H. If $X \subseteq X_i$, then X need no longer be considered. If $X_i \subseteq X$, then X_i needs to be deleted from H. If X has been successfully compared with all $X_i \in H$, then it is added to H. So an important element in our implementation is an efficient method to determine, given two hyperedges, whether one is a subset of the other. We use for that purpose the properties (a) $X \subseteq Y$ iff $X \cap Y = X$ and (b) two bit vectors are equal iff their XOR is zero. Considering these properties, given X and X_i as bit vectors, we need only perform the following bit operations, which are efficiently implementable. Let

$$t = X \wedge X_i.$$

And then $t \oplus X = \emptyset \Rightarrow X \subseteq X_i$, and $t \oplus X_i = \emptyset \Rightarrow X_i \subseteq X$.

Observations (a) and (b) prove the following lemma.

LEMMA 4.3. *The bit operations that are performed in order to keep the maximal refutations are sound.*

5. We consider the complement of each maximal refutation in H, obtaining $H' = \{A \setminus X_1, A \setminus X_2, \ldots, A \setminus X_k\}$.
6. Recalling Section 2, only valid and minimal FDs are to be computed.

Considering $\mathcal{P}(A)\backslash\nu(H')$ and keeping only the minimal FDs, we obtain a hypergraph $H'' = \mu(\mathcal{P}(A)\backslash\nu(H'))$, which is the desired output.

LEMMA 4.4. *H'' is the minimal FDs set with D as the determined attribute and $H'' = \lambda(H')$.*

Proof. Assume $r \models X \rightarrow D$, and that, for no $X' \subset X$, we have $r \models X' \rightarrow D$. Let $Y \in H'$. We claim that $X \cap Y \neq \emptyset$. Because $Y \in H'$, there must exist tuples s and t in r such that $s[B] = t[B]$ $(\forall B \in A \setminus Y)$ and $s[D] \neq t[D]$. But then $s[C] \neq t[C]$ for some $C \in X$, otherwise the dependency $X \rightarrow D$ would be violated. In particular, $C \in Y$. Thus, $X \cap Y \neq \emptyset$, as claimed. The minimality of X means that X belongs to $\lambda(H')$. □

THEOREM 4.1. *The proposed method finds the set of minimal FDs that are present in a relation r.*

Proof. The theorem follows from Lemmata 4.1–4.4. □

Example. Given the set of attributes $A = \{1, 2, 3, 4, 5, 6\}$ and the instance r shown in Fig. 5, we produce the set of FDs that have 6 as its dependent attribute.

We observe that the tuples 1 and 2 produce the refutations $\{1\} \nrightarrow 6$, $\{2\} \nrightarrow 6$ and $\{1, 2\} \nrightarrow 6$; tuples 1 and 3 produce the refutations $\{2, 4\} \nrightarrow 6$, and so on. Keeping only the maximal refutations, we obtain hypergraph $H = \{\{1, 2\}, \{2, 4\}\}$ in the first step of the method. Next, we obtain the hypergraph of the complements of the hyperedges in H, which is $H' = \{\{3, 4, 5\}, \{1, 3, 5\}\}$. Finally, the operator λ produces the FDs in minimal form: $H'' = \lambda(H') = \{\{1, 4\}, \{3\}, \{5\}\}$. That is, we have discovered the FDs $\{1, 4\} \rightarrow 6$, $\{3\} \rightarrow 6$ and $\{5\} \rightarrow 6$.

Note that the FDs that are found in a particular instance of a relation may not be valid in the real world in general. That is, they may arise from coincidences in the data present in the particular instance that has been used. Therefore, in a real-world situation the FDs that are output by our method (or by any other method for FD mining) need to be confirmed by experts or by users. Our conjecture is that the amount of FDs that stem from coincidences can be high in general, since there are many conditions in the data that can be a product of circumstances and not of a general rule. But these FDs are not *incorrect* from a FD mining perspective. In fact, the *interpretation* that is given to the data depends on the semantics of the data from the user's or the business model's perspective and corresponds to the area of software engineering. Indeed, one could imagine the same syntactical data being interpreted in many different ways, depending on its semantics.

4.2. Implementation

For the computational manipulation of hypergraphs we used boolean matrices, as defined in Section 3.1, where each row corresponds to an hyperedge $X_i \in H'$ defined over the base set $A = \{a_1, \ldots, a_{|A|}\}$, where $i = 1, \ldots, |\ H'\ |$ and $j = 1, \ldots, |\ A\ |$. The following matrix shows an example of a representation

Tuple ID	1	2	3	4	5	6
1	a_1	b_3	c_2	d_1	e_4	f_1
2	a_1	b_3	c_3	d_3	e_1	f_2
3	a_2	b_3	c_5	d_1	e_5	f_4
4	a_3	b_3	c_2	d_3	e_3	f_1
5	a_4	b_2	c_2	d_8	e_2	f_1
6	a_5	b_4	c_4	d_1	e_3	f_1
7	a_1	b_1	c_3	d_7	e_6	f_2

FIGURE 5. Relation instance with $A = \{1, 2, 3, 4, 5, 6\}$.

of a hypergraph H':

$$H' = \begin{array}{c} \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}.$$

The main data structures used in the implementation of the algorithms are trees and array lists, among others. Also, for the codification of the attribute values of the input relation instance we used hash tables.

Next, Algorithm 5 is the implementation of the proposed method. Procedure REFS-MAX obtains all the maximal FD refutations for all the elements in A as the dependent attribute. Next, the complement of each refutation is computed and finally the FDs are obtained by means of the procedure TRANSV, which computes the minimal transversals and depends on the algorithm that is used for hypergraph dualization: either Berge's algorithm, or Kavvadias and Stavropoulos', or Murakami and Uno's (with the DFS Algorithm).

5. EXPERIMENTAL RESULTS

We will call our tool FSGS in this section. We carried out a set of experiments in order to evaluate the performance of the approach that we propose and to compare it with existing tools. We used several relation instances from real-world databases, which were mainly obtained from the *UCI repository of machine learning databases* [23]. Experiments were carried out on a Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz with 4 GB RAM, running Debian GNU/Linux 6.0.

We first made some experiments to determine which algorithm is the best for the computation of operator λ, the minimal transversals. Table 1 shows the performance of algorithm FSGS using the three considered algorithms for the computation of λ: that of Berge, that of Kavvadias and Stavropoulos, and that of Murakami and Uno. We observe that the algorithm by Murakami and Uno has the best performance among the three, so this is the one we will use for the purpose of the subsequent experimentation.

Next, we performed experiments with real-world instances using the algorithms TANE and FSGS (with the algorithm by Murakami and Uno for the computation of minimal transversals), obtaining all the minimal FDs from the relations. Table 2 shows the results obtained for 14 instances considering different numbers of tuples in the relation instance $|r|$ and different numbers of attributes $|R|$. The response time was measured for both algorithms, considering that of course both found the same minimal FDs.

From this experimentation it can be concluded that the algorithm TANE shows a good performance, outputting the results in a reasonable time only when the number of attributes is <30 approximately. Remember that this algorithm performs a search over the boolean lattice shown in Fig. 2. When working with instances with a larger number of attributes, TANE produces a collapse of the computer's main memory. However, with a small number of attributes TANE's performance is good, even with a large number of tuples in the relation instance.

On the other hand FSGS produced a result in all the performed experiments, considering a large number of attributes and of tuples. Although the execution time of course increases with the size of the instances, it is noteworthy that it always output the correct results.

To perform more detailed experiments with TANE and FSGS, we used the instance of the relation FopTrain [23] and we obtained the FDs varying the number of attributes for the instance with 500 tuples (Table 3) and for the instance with 1000 tuples (Table 4). It can be observed in these experiments

Algorithm 5. FSGS.

1. $refs :=$ call REFS-MAX(r)
2. $compRefs := \emptyset$
3. **for each** $H \in refs_i$ **do**
4. $\quad H' := \emptyset$
5. $\quad$ **for each** $X \in H_i$ **do**
6. $\quad\quad H' := H' \cup X^C$
7. $\quad$ **end for**
8. $\quad compRefs := compRefs \cup H'$
9. **end for**
10. **for each** $H' \in compRefs_i$ **do**
11. $\quad FDs := FDs \cup TRANSV(H')$
12. **end for**

TABLE 1. FSGS with different algorithms for the computation of minimal transversals.

Instance	$\|r\|$	$\|R\|$	#FDs	FSGS-BERGE (s)	FSGS-KS (s)	FSGS-MU (s)
Particle	3302	50	4350	810.0	285.0	244.0
Parkinson's Telemonitoring	5875	26	855	470.0	381.0	195.0
Automobile	205	20	494	54.0	11.0	3.0
Hepatitis	155	19	1250	13.0	4.0	1.0
Wine	178	13	61	1.0	0.5	0.1

TABLE 2. Performance of the algorithms for FD extraction.

Instance	$\|r\|$	$\|R\|$	#FDs	TANE (s)	FSGS (s)
Wine	178	14	1374	0.1	0.1
Hepatitis	155	20	8250	5.0	1.0
Automobile	205	26	4176	18.0	3.0
Waveform	5000	22	24 002	3.0	148.0
Parkinson's Telemonitoring	5875	22	10 066	4.0	195.0
Flag	178	31	427 115		19.0
Dermatology	366	35	1 609 865		107.0
KRVSKP	3198	37	7177		47.0
SPECTF	80	45	506 246		48.0
Particle	3302	50	4350		244.0
Spambase	4601	58	12 126 834		5131.0
Bands	550	40	1 593 147		581.0
Sonar	208	61	100 631		3.0

TABLE 3. Behavior of the algorithms with an instance, varying the number of attributes.

Instance	$\|r\|$	$\|R\|$	#FDs	TANE (s)	FSGS (s)
$FopTrain_1$	500	10	58	0.6	0.1
$FopTrain_2$	500	15	496	0.9	0.5
$FopTrain_3$	500	20	1224	1.3	2.1
$FopTrain_4$	500	25	2715	3.0	5.3
$FopTrain_5$	500	30	6074	11.3	9.0
$FopTrain_6$	500	35	13 907	720.0	19.0
$FopTrain_7$	500	40	36 330		28.0
$FopTrain_7$	500	45	99 924		43.0
$FopTrain_7$	500	50	322 856		88.0

TABLE 4. Behavior of the algorithms with an instance, varying the number of attributes.

Instance	$\|r\|$	$\|R\|$	#FDs	TANE (s)	FSGS (s)
$FopTrain_1$	1000	10	99	1.0	1.0
$FopTrain_2$	1000	15	566	1.5	3.0
$FopTrain_3$	1000	20	1614	2.1	9.6
$FopTrain_4$	1000	25	3663	6.5	17.0
$FopTrain_5$	1000	30	7826	24.0	33.0
$FopTrain_6$	1000	35	20 040	780.0	72.0
$FopTrain_7$	1000	40	59 868		111.0

that, as it was the case in the previous ones, TANE overflows and ceases to produce an output at ∼30 attributes. On the other hand, FSGS increases its response time but is able to produce an output in reasonable times in all the cases. In Fig. 6, we show the graph for the experiments in Table 3, and in Fig. 7 the graph for those in Table 4.

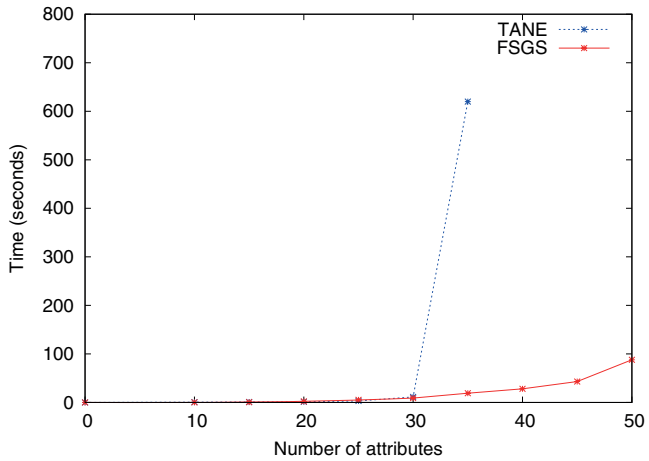

FIGURE 6. Behavior of the experiments from Table 3.

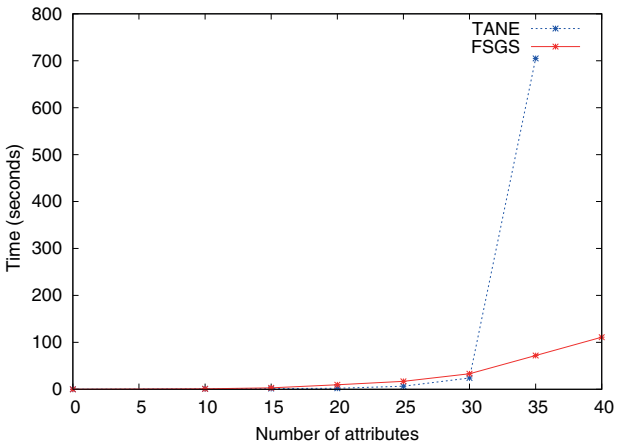

FIGURE 7. Behavior of the experiments from Table 4.

Finally, and in order to understand the internal behavior of FSGS, we measured the time that our algorithm spends in its two main steps: (a) the computation of the refutations and (b) the computation of the minimal transversals. We used the instance of the relation *SpamBase* [23] considering:

(i) *Variation of the number of attributes while fixing the number of tuples*: Fig. 8 shows the behavior of FSGS considering the instance with 3301 tuples, varying the number of attributes up to 58. It can be observed that the computation of the refutations takes most of the total time, but as the number of attributes increases, the computation time for the minimal transversals also increases.

(ii) *Variation of the number of tuples while fixing the number of attributes*: Fig. 9 shows the behavior of FSGS considering the instance with 40 attributes and varying the number of tuples up to 3301. It can be observed that the computation of the refutations takes most of the total time and that this is so while augmenting the number of tuples. The time consumed by the computation of the minimal transversals is also augmented, but to a less extent.

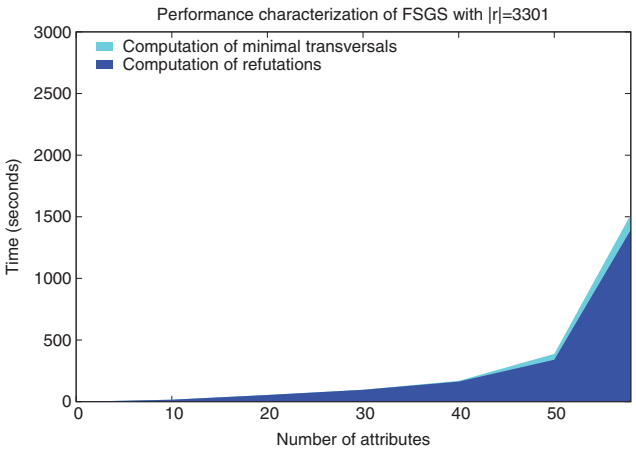

FIGURE 8. FSGS varying the number of attributes while fixing the number of tuples.

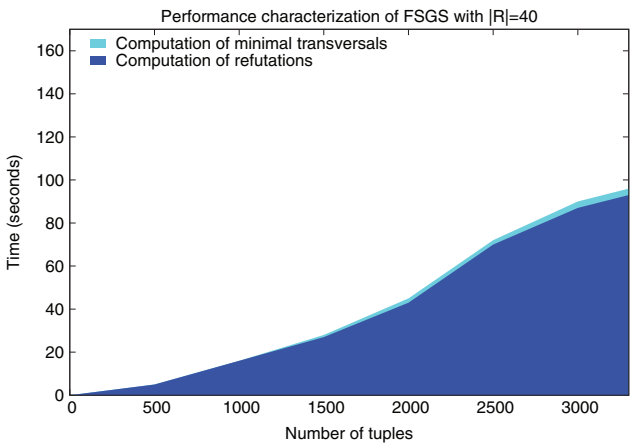

FIGURE 9. FSGS varying the number of tuples while fixing the number of attributes.

Figures 8 and 9 show the effectiveness of our proposal, and it is possible to observe that the process of obtaining the refutations, a quadratic algorithm, can take much more time than the computation of the minimal transversals, which is not polynomial. This is due to the fact that the output of the first process is relatively small, and therefore the second process has a smaller input. This shows the advantages of discovering FDs starting from refutations.

6. CONCLUSIONS AND FUTURE WORK

In this work, we reviewed the main tools for the discovery of FDs in database instances, highlighting the algorithm TANE by Huhtala *et al.* [8], which is one of the better known ones. We showed that the efficiency of this tool decreases as the number of attributes in the relation is augmented, making its use very difficult for relations with more than 30 attributes.

The new algorithms available for the hypergraph duality problem and the idea of refutations (or antikeys [6]) allowed us to develop an efficient tool to obtain the FDs present in an instance of a relation based on computing the refutations for FDs and then applying the hypergraph operators that compute the minimal transversals. The experimentation that we performed, using the algorithm by Murakami and Uno for the computation of the minimal transversals, shows that our method works well, even with relations with a large number of attributes. Since our algorithm is based on the computation of minimal transversals, it will benefit from the algorithms that can be proposed in the future for this problem.

As for future work, we plan to implement more algorithms for the computation of minimal transversals.

FUNDING

For the work reported here, I.D.S. was supported in part by the Fulbright Specialist Program. J.F. was supported by DIUBB project 143119 2/I. G.G. was supported by Mecesup project UBB0704, Chile. This work was carried out under the sponsorship of the *Laboratorio de Base de Datos*, Universidad del Bío-Bío, Chile.

REFERENCES

[1] Calvanese, D., De Giacomo, G. and Lenzerini, M. (2001) Identification Constraints and Functional Dependencies in Description Logics. *International Joint Conference on Artificial Intelligence*, Seattle, USA, pp. 155–160.

[2] Lin, C. (2008) Migrating to relational systems: problems, methods, and strategies. *Contemp. Manag. Res.*, **4**, 369–380.

[3] Eiter, T., Makino, K. and Gottlob, G. (2008) Computational aspects of monotone dualization: a brief survey. *Discrete Appl. Math.*, **156**, 2035–2049.

[4] Fredman, M. and Khachiyan, L. (1996) On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, **21**, 618–628.

[5] Mannila, H. and Räihä, K.-J. (1994) Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng.*, **12**, 83–99.

[6] Eiter, T. and Gottlob, G. (1991) Identifying the Minimal Transversals of a Hypergraph and Related Problems. Technical Report CD-TR 91/16. Christial Doppler Labor Für Expertensyteme, Technische Universität Wien.

[7] Yao, H. and Hamilton, H. (2008) Mining functional dependencies from data. *Data Min. Knowl. Discov.*, **16**, 197–219.

[8] Huhtala, Y., Karkkainen, J., Porkka, P. and Toivonen, H. (1999) Tane: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, **42**, 100.

[9] Novelli, N. and Cicchetti, R. (2001) Fun: An Efficient Algorithm for Mining Functional and Embedded Dependencies. *Database Theory—ICDT 2001*, London, UK, pp. 189–203. Springer.

[10] Flach, P. and Savnik, I. (1999) Database dependency discovery: a machine learning approach. *AI Commun.*, **12**, 139–160.

[11] Lopes, S., Petit, J. and Lakhal, L. (2000) Efficient Discovery of Functional Dependencies and Armstrong Relations. *Advances in Database Technology—EDBT 2000*, Konstanz, Germany, pp. 350–364. Springer, Berlin, Heidelberg.

[12] Wyss, C., Giannella, C. and Robertson, E. (2001) Fastfds: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances. *Data Warehousing and Knowledge Discovery*, Munich, Germany, pp. 101–110. Springer, Berlin, Heidelberg.

[13] Distel, F. and Sertkaya, B. (2011) On the complexity of enumerating pseudo-intents. *Discrete Appl. Math.*, **159**, 450–466.

[14] Baixeries, J. (2004) A Formal Concept Analysis Framework to Mine Functional Dependencies. *Workshop on Mathematical Methods for Learning*, Como, Italy.

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

[16] Yao, H., Hamilton, H. and Butz, C. (2002) Fd_mine: Discovering Functional Dependencies in a Database using Equivalences. *ICDM*, Maebashi City, Japan, pp. 729–732. IEEE Computer Society.

[17] Berge, C. (1989) *Hypergraphs*. Elsevier Science Publishers B.V., Amsterdam.

[18] Polyméris, A. (2008) Stability of two player game structures. *Discrete Appl. Math.*, **156**, 2636–2646.

[19] Eiter, T. and Gottlob, G. (2002) Hypergraph Transversal Computation and Related Problems in Logic and AI. In Flesca, S., Greco, S., Ianni, G. and Leone, N. (eds), *Logics in Artificial Intelligence*, Lecture Notes in Computer Science 2424, pp. 549–564. Springer.

[20] Kavvadias, D. and Stavropoulos, E. (2005) An efficient algorithm for the transversal hypergraph generation. *J. Graph Algorithms Appl.*, **9**, 239–264.

[21] Murakami, K. and Uno, T. (2013) Efficient Algorithms for Dualizing Large-Scale Hypergraphs. *ALENEX*, New Orleans, USA, pp. 1–13.

[22] Khachiyan, L., Boros, E., Elbassioni, K. and Gurvich, V. (2006) An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Appl. Math.*, **154**, 2350–2372.

[23] Bache, K. and Lichman, M. (2013) UCI machine learning repository. http://archive.ics.uci.edu/ml/citation_policy.html