Inteligencia Artificial, 24(68) (2021), 37-52
doi: 10.4114/intartif.vol24iss68pp37-52

INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

KeyFinder: An Efficient Minimal Keys Finding Algorithm For Relational Databases

Moussa Demba
Dept. of Quantitative Mathematics and Computer Science
ISCAE- Higher Institute of Accounting and Business Administration, Nouakchott, Mauritania.
bah.demba@iscae.mr

Abstract In relational databases, it is essential to know all minimal keys since the concept of database normalization is based on keys and functional dependencies of a relation schema. Existing algorithms for determining keys or computing the closure of arbitrary sets of attributes are generally time-consuming. In this paper we present an efficient algorithm, called *KeyFinder*, for solving the key-finding problem. We also propose a more direct method for computing the closure of a set of attributes. *KeyFinder* is based on a powerful proof procedure for finding keys called tableaux. Experimental results show that *KeyFinder* outperforms its predecessors in terms of search space and execution time.

Keywords: Attribute Closure, Minimal keys, Functional dependencies, Inference system, relational databases.

1 Introduction

In database Design, it is vital to know all minimal keys for a relation schema R since all normal forms, except for 1NF, are based on key constraints. Therefore, finding keys is indispensable for a database designer. The efficient discovery of all keys from a set of functional dependencies remains a challenge in database research although many algorithms have been proposed. A relation schema R is in second normal form (2NF) if every non-prime attribute (attribute that is not involved in any minimal key) is fully functionally dependent on every minimal key of R. A relation R satisfies the third normal form (3NF) if for any nontrivial functional dependency (FD) $X \rightarrow Y$, either X is a super key or Y consists of prime attributes (attributes which are parts of some minimal keys) [10]. This is the point where a database designer has to know all keys of a relation schema. It has been shown that the number of minimal keys (also called candidate keys) for a relational system can be factorial in the number of FDs or exponential in the number of attributes [18].
H. Saiedian et al. have presented an algorithm for finding all minimal keys of a relational schema [23]. In their approach, the FDs among the attributes in a database table are represented as a graph. However, their algorithm can find all the minimal keys only if the FDs' graph is not strongly connected.The algorithm proposed in [17] derives only one minimal key. Bahmani et al. [2] have proposed an automatic method based on graph theory for key generation. In Bardoloi et al. [5], a generalized graph-based approach to find all minimal keys in a relational database schema is proposed. Their approach cannot find all minimal keys (see their example 3, they compute only one minimal key out of three).
Recently, Fernandez et al. [12] proposed a method for finding all minimal keys using strategic port graph Rewriting. Walst proposed a first approach based on tableaux [26] to compute minimal keys. However,

his approach is only suitable for small data and require functional dependencies to be in a canonical form, i.e. Horn clauses. Cordero et al. [7, 8] proposed a tableaux-like method called *SST* to infer all minimal keys. Their approach is also based on tableaux and can infer all minimal keys but with duplicated keys. They also proposed another parallel method, named *Closure Keys*, to improve the *SST* algorithm [3].
The problem we are interested in consists to find, given a set of functional dependencies Γ and a set of all attributes Ω, all the *minimal* sub-set $K \subseteq \Omega$ such that the FD $K \to \Omega$ holds in Γ. K is called a *minimal key*. The proposed algorithm for key-finding, called *KeyFinder*, is inspired from the tableaux-based methods in [26, 8]. *KeyFinder* shows better performance than its predecessors not only in terms of search space (number of nodes of a tableaux) but also in terms of execution time.
The rest of this paper is organized as follows: In Section 2, we give some preliminary notions related to the key-finding problem. We will focus on works based on inference systems that are most related to our work. Section 3 presents our proposed algorithm *KeyFinder*. Section 4 shows some experimental results and conclude in Section 5.

2 Background

From now on, we assume that Ω is a finite set of attributes, Γ is a set of FDs, capital letters A, B, C,... are subset of Ω and small letters a, b, c, ... are attributes. Moreover, A-B (resp. AB) denotes the set difference operator (resp. the union operator). A relational database schema $R(\Omega, \Gamma)$ consists of a finite set of attributes and a finite set of FDs over Ω.

2.1 Basic Definitions

Definition 2.1. *Let X and Y be two subset of attributes. We say that X functionally determines Y, written $X \to Y$, iff the value of X determines a unique value for Y, i.e. for any two different tuples t_i and t_j if $t_i[X] = t_j[X]$ then $t_i[Y] = t_j[Y]$. The determinant X is called the FD's left-hand side (or lhs) and the dependent Y is called the FD's right-hand side (or rhs).*

Definition 2.2. *The closure of an attribute set $X \subseteq \Omega$, written X^+, w.r.t a set of FDs Γ is the set of all attributes Y for which the FD $X \to Y$ holds in Γ, i.e. $X^+ = \{Y | \Gamma \models X \to Y\}$. Algorithm 1 is the standard one used for computing the closure of an arbitrary set of attributes.*

Algorithm 1: Standard Closure Algorithm.

```
Input: $\Gamma$, X.
Output: $X^+$
1 $X^+$=X
2 while changes to $X^+$ do
3     foreach $A \to b \in \Gamma$ do
4         if ($A \subseteq X^+$ & $b \notin X^+$) then
5             $X^+ = X^+ \cup \{b\}$
```

Definition 2.3. *The closure of a set of FDs Γ, Γ^+ for short, is the set of all FDs logically implied by Γ, $\Gamma^+ = \{X \to Y | \Gamma \models X \to Y\}$.*

Definition 2.4. *Let $R(\Omega, \Gamma)$ be a relational database schema. A subset of attributes $\mathcal{K} \subseteq \Omega$ is a super key of* R *if $\Gamma \models (\mathcal{K} \to \Omega)$, i.e. $\mathcal{K}^+ = \Omega$. If there is no $\mathcal{K}' \subset \mathcal{K}$ such that $\Gamma \models (\mathcal{K}' \to \Omega)$, then $\mathcal{K}$ is called a minimal key. If $R(\Omega, \Gamma)$ has more than one minimal key, one of them is chosen by the designer as the* primary key.

It is well known that an FD $X \to Y$ is said to be *implied* by another set of FDs Γ, written $\Gamma \models X \to Y$, if any relation that satisfies Γ also satisfies $X \to Y$. The set of all implied FDs can be obtained using the Armstrong's inference rules [1]:

(R1) Inclusion: if $Y \subseteq X$ holds, then $X \to Y \in \Gamma$;

(R2) Augmentation: $(X \to Y \in \Gamma) \Rightarrow (XZ \to Y \in \Gamma)$, for any $Z \subseteq \Omega$; and

(R3) Transitivity: $(X \to Y \in \Gamma,\ Y \to Z \in \Gamma) \Rightarrow (X \to Z \in \Gamma)$

Other rules have been derived from the above axioms, such as:

(R4) Union: $(X \to Y \in \Gamma,\ X \to Z \in \Gamma) \Rightarrow (X \to YZ \in \Gamma)$.

(R5) Composition: $(X \to Y \in \Gamma,\ U \to V \in \Gamma) \Rightarrow (XU \to YV \in \Gamma)$.

These inference rules can be used to infer all FDs that are implied by a given set of FDs. However, it is not easy to use these rules to check whether a given FD can be derived from a set of FDs. This difficulty makes the *key-finding problem* more complicated because of the size of its search space. To deal with this problem, R. Wastl [26] proposed, for the first time, an inference system (called $\mathbb{K}$) based on Tableaux for computing keys. However, the $\mathbb{K}$ system requires the FDs to be Horn clauses which increases the search space. Later, P. Cordero et al. [8] extended the $\mathbb{K}$ system to a more powerful method, called *SST*, to deal with more general formulas and then reduce the search space. The *SST* method is based on the following two inference rules: *Strong Simplification* (*sSimp*) and *Left Simplification* (*lSimp*):

$$\frac{A \to B \quad C \to D}{A(C-B) \to (D-AB)}(sSimp) \qquad\qquad \frac{A \to B \quad C \to D}{A(C-B) \to D}(lSimp)$$

i.e. the FDs at the top imply the FD at the bottom. The rule *sSimp* (resp. *lSimp*) is used to compute new FDs (resp. child nodes of a Tableaux). Each node is obtained from Γ by applying n-1 times the *lSimp* rule. The starting point is the root node labeled with the input data Ω and Γ. Next, the process continues by applying the sSimp rule over a selected FD $X \to Y \in \Gamma$ and $\Gamma - \{X \to Y\}$. At the end of the search, all minimal keys appear, at least once, in the leaves.
However, before applying any method for finding keys it's useful to enumerate all attributes that must be in any key. Those attributes, called the *core* attributes, represent the intersection of all keys [15, 8], i.e. the attributes not appearing in any rhs of Γ. The *body* corresponds to those attributes that may be part of keys. Fig.1 shows the *core* and the *body* of a set of attributes Ω where K_1 and K_2 are supposed to be the only minimal keys of R. Note that, the attributes that occur only at the rhs of FDs will not occur in any key.

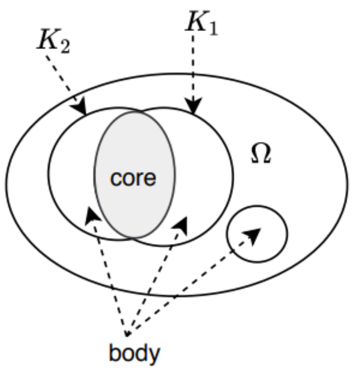


Figure 1: Core and Body.

In [8] the authors proposed an efficient method for computing the *core* and the *body* of a set of FDs Γ.

Definition 2.5. *The core and the body of* Γ *are defined as follows [8]:*
$core = \Omega - (\bigcup_{X \to Y \in \Gamma} Y)$ *and* $body = (\bigcup_{X \to Y \in \Gamma} X) - core^{+}$

Example 2.1. *Let* $\Omega = abcdefgh$ *and* $\Gamma = \{c \to ad, e \to fg, d \to f, h \to e, g \to h\}$*. We have* $core = bc$ *and* $body = egh$*. Therefore, the set of keys* $\mathcal{K} = \{bce, bch, bcg\}$*. Note that the attribute* b *does not appear in* Γ *and FDs are not necessary Horn clauses.*

Theorem 2.1. *Let $\mathcal{K}$ be the set of all minimal keys of R. We have the following results:*

(a). $\forall k \in \mathcal{K}$, $core \subseteq k \subseteq (core \cup body)$,

(b). *if* $body = \emptyset$, *then* $\mathcal{K} = core$, *and*

(c). $core^+ \cap body = \emptyset$.

Lemma 2.1. *For every attribute $a \in \Omega$, if a is a prime attribute then $a \in (core \cup body)$.*

Proof. Let a be a prime attribute, i.e. there is a key $\mathcal{K}$ s.t a is part of $\mathcal{K}$. From corollary 2.1.a, we can say that $a \in (core \cup body)$. □

Lemma 2.2. *If $core^+ = \Omega$ then core is the only key of R.*

Proof. Suppose K is a minimal key of R. By theorem 2.1.a, we have $core \subseteq K$. Since $core^+ = \Omega$ and K is a minimal key (not a super key), we must have $core$=K. □

Now, the *key-finding problem* for a relational schema $R(\Omega, \Gamma)$ is reduced to the *key-finding problem* for $R(\Omega', \Gamma')$ where $\Omega' = body$ and $\Gamma' = \{X \cap \Omega' \to Y \cap \Omega' | X \to Y \in \Gamma\}$. Note that, in Γ' most of the *non-prime* attributes are removed. If $\mathcal{K}$ is a key for $R(\Omega', \Gamma')$, then $(\mathcal{K} \cup core)$ is a key for $R(\Omega, \Gamma)$, see [7] for more details. However, this reduction is fundamentally important if $core \neq \emptyset$.
SST implements another method called SL_{FD} closure for computing attribute closures. For a more detailed description of the SL_{FD} closure method we refer the reader to [21]. Here is the algorithm *SST*.

Algorithm 2: Reduction method

```
Input: Ω, Γ.
Output: The set $\mathcal{K}$ of all minimal keys.
1 $\Omega' = body(\Omega, \Gamma)$    // use the $SL_{FD}$ closure algorithm in [21] to compute $core^+$
2 $\Gamma' = \{X \cap \Omega' \to Y \cap \Omega' | X \to Y \in \Gamma\}$
3 $\mathcal{K}' = Tableaux(\Omega', \Gamma')$
4 $\mathcal{K} = \{k \cup core | k \in \mathcal{K}'\}$
```

Algorithm 3: $Tableaux(\Omega, \Gamma)$

```
Input: Ω, Γ.
Output: The set $\mathcal{K}$ of all minimal keys.
    • The root node of the tree will be $(\Omega, \Gamma)$.
    • For each node $(K, \Gamma)$, including the root one, and for each minimal FD $A \to B \in \Gamma$, a new child
      node $(K', \Gamma')$ is added where
        – The edge connecting the parent node $(K, \Gamma)$ to its successor node $(K', \Gamma')$ is labeled with the
          FD $A \to B$.
        – $K' \to \Omega$ is the result of $lSimp(A \to B, K \to \Omega)$.
        – $\Gamma' = \Gamma' \cup sSimp(A \to B, C \to D)$ for every $C \to D \in \Gamma$.
        – If $\Gamma' = \emptyset$, then $K'$ is a leaf i.e. $K'$ is a key and $\mathcal{K} = \mathcal{K} \cup \{K'\}$, else
        – call $Tableaux(K', \Gamma')$.
    • Remove any non minimal FD from $\mathcal{K}$.
    • the method returns $\mathcal{K}$.
```

2.2 Examples

We present two examples to illustrate the *SST* algorithm.

Example 2.2. *(From [3]) Let $R(\Omega, \Gamma)$ be a relation schema with $\Omega = abcdeg$ and $\Gamma = \{c \to a,\ d \to eg,\ ab \to c,\ bc \to d,\ be \to c,\ ce \to g,\ cg \to b\}$. In this case, core $= \emptyset$ and body $= \Omega$. The SST method is then called with Ω and Γ as input. Fig.2 shows the SST method in action. As we can see, there are 21 nodes and 7 keys with one redundant key, i.e. the key* cd *is calculated twice.*

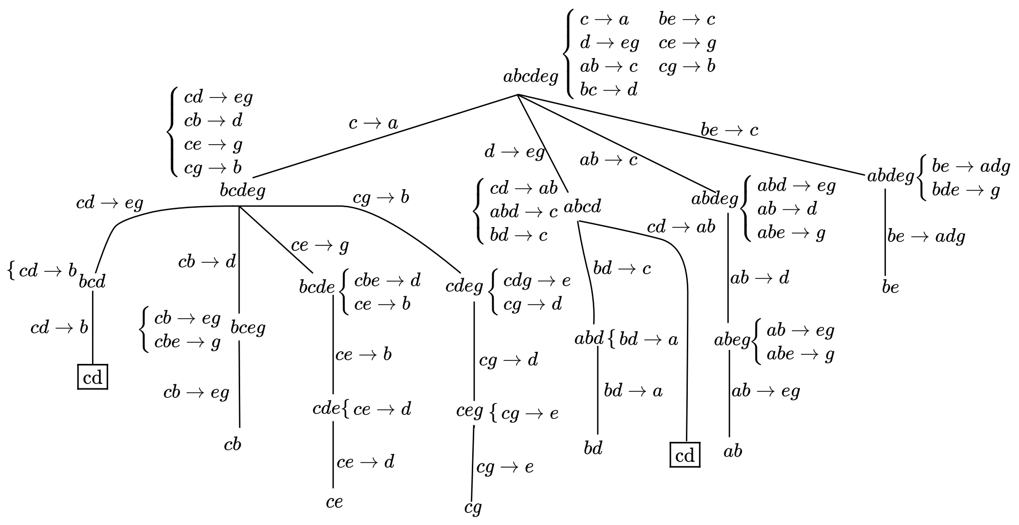


Figure 2: Illustration of the SST method on Example 2.2.

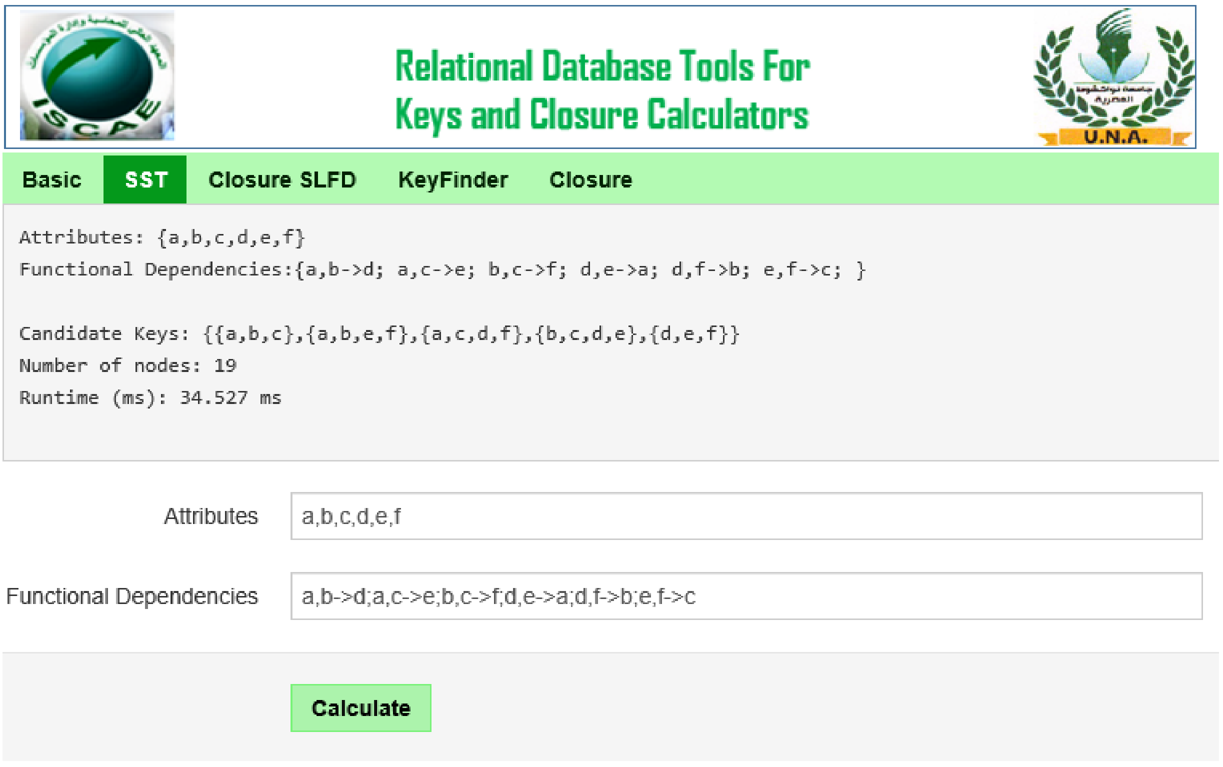
Let's consider another example to show the weaknesses of the *SST* method.

Example 2.3. *Let $R(\Omega, \Gamma)$ be a relation schema with $\Omega = abcdef$ and $\Gamma = \{ab \to d,\ ac \to e,\ bc \to f,\ de \to a,\ df \to b,\ ef \to c\}$. As in the previous example, core $= \emptyset$ and body $= \Omega$. The corresponding screenshot is given in Fig.3. There are 19 nodes and 5 minimal keys. In its original implementation [8], SST computed 7 redundant keys.*

From the two examples, we can see that some keys are duplicated. The authors recognize that preventing the computation of redundant keys and to open extra-branches are not a trivial issue. Figures 2 and 3 show that their *inclusion-minimality strategy* cannot prevent the computation of redundant keys. The authors proposed in [3] another parallel algorithm named *SLD Keys* for reducing the search space of the *SST* algorithm. With this latter, However, the problem of duplicated keys remain unsolved.
In the next section, we will propose a new algorithm for characterizing branches leading to duplicated keys and prevent as much as possible useless computations, i.e. computation of non-minimal keys.

3 A new Algorithm for Discovering Keys

Let's consider the example in Fig.2, the set of FDs used for computing the key cg is $\{c \to a,\ cg \to b,\ cg \to d,\ cg \to e\}$. Applying the rules of *union* and *composition* (see section 2.1) on this set of FDs yields the FD $cg \to abde$, i.e. the nontrivial closure of cg, see definition 3.1. This observation holds also for the other branches. Therefore, to reduce the size of the tree, one can determine the closure of some attribute sets before applying the rule of *sSimp*. The central idea of *KeyFinder* is (i) to reduce the size of Γ, if possible, (ii) to compute the closure of some attributes (those that are relevant for key computations) and (iii) to remove all redundant or implied FDs. The proposed algorithm uses one single derivation rule for determining the closure of an attributes' set. Before presenting the proposed algorithm, we need to present some concepts related to the closure of a set of attributes.



```
Attributes: {a,b,c,d,e,f}
Functional Dependencies:{a,b->d; a,c->e; b,c->f; d,e->a; d,f->b; e,f->c; }

Candidate Keys: {{a,b,c},{a,b,e,f},{a,c,d,f},{b,c,d,e},{d,e,f}}
Number of nodes: 19
Runtime (ms): 34.527 ms
```

Attributes: a,b,c,d,e,f

Functional Dependencies: a,b->d;a,c->e;b,c->f;d,e->a;d,f->b;e,f->c

Figure 3: Screenshot of the SST method on Example 2.3.

3.1 The Closure of Relevant Attributes

We are going to present a new approach, not based on Armstrong's rules, for computing attributes closure. The proposed approach substantially reduces the computational complexity related to the closure-finding problem.

Definition 3.1. *A nontrivial closure of an attribute set $A \subseteq \Omega$, written A^*, is defined as $A^* = A^+ - A$ where A^+ is the closure of A.*

Several algorithms exist in the literature for computing attribute closure [11, 24, 10, 21, 17]. However, all the existing algorithms, except [20] improved in [21] and named SL_{FD}, are governed by the Armstrong's inference rules. In [20, 21] the authors proposed, for the first time, a linear closure algorithm based on three inference rules. However, finding A^* from A^+ using existing algorithms is very costly algorithmically. Here we propose a more direct computational method for computing A^* without determining A^+. The proposed method consists of one single derivation rule, named *closure*: Let $\Gamma = \{A \to B, C \to D\}$ be a set of FDs where $C \subseteq AB$, the nontrivial closure A^* is obtained by repeatedly applying the following rule:

$$closure(A) = \begin{cases} B(D - AB) & and\ \Gamma = \Gamma - \{C \to D\} & \text{if } C \subseteq AB \\ B & and\ \Gamma = \Gamma - \{C \to D\} & \text{if } D \subseteq AB \\ B & and\ \Gamma = \Gamma - \{C \to D\} \cup \{(C - AB) \to (D - AB)\} & \text{otherwise} \end{cases}$$

In this case, we are able to calculate the non-trivial attributes closure in one iteration. Initially $B = \emptyset$ and $A \to \emptyset$ is known to be an axiom. A^* is obtained by using the closure rule until no dependency can be applied. This derivation rule is simple, easy to understand and can be used to compute the nontrivial closure of any set of attributes. This rule can also be used to compute $A^+ = A \cup A^*$.

Example 3.1. *Consider the set of FDs* $\Gamma = \{ab \to c, a \to b, bc \to a, ac \to d, deh \to f, ef \to g, ag \to e\}$. *Suppose we want to compute the nontrivial closure of adg, i.e.* $(adg)^*$ *w.r.t to* Γ. *Fig.4 shows our closure algorithm in action step by step for computing* $(adg)^*$ *and Fig.5 corresponds to the program output. The algorithm returns* $(adg)^* = bec$.

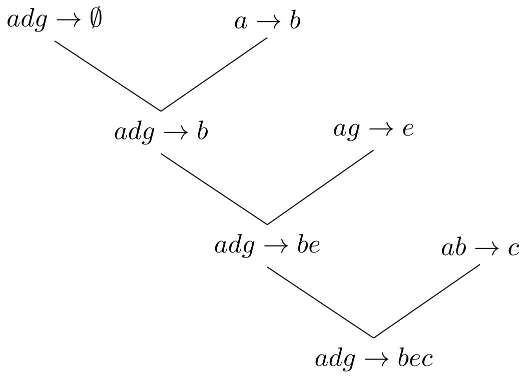
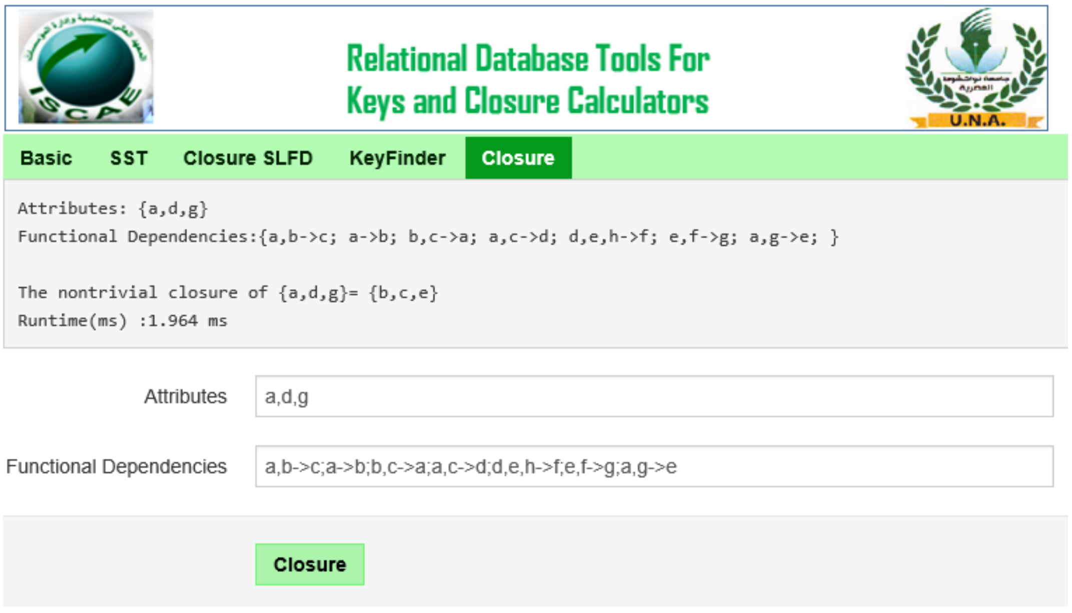


Figure 4: Computing $(adg)^*$

In the literature, the closure of a set of FDs is obtained by repeatedly applying Armstrong's rules [1]. Closure operator is a key element in FCA landscape [13, 16]. Here, we will show that using the rule of *closure* one can calculate the nontrivial closure of Γ^* of Γ. A systematic way to determine Γ^* is to find the nontrivial closure of any attribute set A that appears at the lhs of an FD. For example, considering the example 3.1 we have Γ^*=$\{ab \to cde,\ a \to bcde,\ ac \to ebd,\ f \to abcde\}$. Note that, we are not interested in determining the *minimal cover* of Γ as this notion requires the right side of every FD to be a single attribute [19, 25].



```
Attributes: {a,d,g}
Functional Dependencies:{a,b->c; a->b; b,c->a; a,c->d; d,e,h->f; e,f->g; a,g->e; }

The nontrivial closure of {a,d,g}= {b,c,e}
Runtime(ms) :1.964 ms
```

Attributes	a,d,g
Functional Dependencies	a,b->c;a->b;b,c->a;a,c->d;d,e,h->f;e,f->g;a,g->e

Figure 5: Screenshot for computing $(adg)^*$.

Definition 3.2. *An FD* $A \to B$ *implies another FD* $C \to D$ *iff* $A \subseteq C$ *and* $D \subseteq B$.

Definition 3.3. *An FD $C \to D$ is implied by a set of FDs Γ iff there is another FD $A \to B \in \Gamma$ such that $A \to B$ implies $C \to D$, i.e. $\Gamma \models C \to D$.*

Example 3.2. *Let $\Gamma = \{bg \to ahdc, aeh \to bcdg, gaf \to bch, gbe \to ac\}$. The FD $gbe \to ac$ is implied by Γ as $bg \to ahdc$ implies $gbe \to ac$. It is clear that implied FDs are redundant ones.*

Corollary 3.1. *Let Γ be a set of FDs, $A \to B$ an FD implied by Γ and $\Gamma' = \Gamma - \{A \to B\}$. Thus Γ and Γ' are functionally equivalent, i.e. $\Gamma^+ = \Gamma'^+$.*

Algorithm 4: Algorithm impliedFD($C \to D$, Γ).

```
Input: an FD set $\Gamma$ and an FD $C \to D$.
Output: true or false
1 foreach $A \to B \in \Gamma$ do
2     if $(A \subseteq C)$ AND $(D \subseteq B)$ then
3         return true
4 return false
```

To reduce further the search space, i.e. the size of a tableaux in our case, we propose a new pruning strategy. Removing unnecessary FDs will optimize the computation of closure. Indeed, the number of iterations for computing the closure is related to the number of dependencies.

Definition 3.4. *Let $A \to B$ and $C \to D$ be two non-implied FDs with $A \subseteq C$, $A \cap B = C \cap D = \emptyset$ and $D \not\subseteq B$. We have the following rule:*

$$\frac{A \to B \quad C \to D \quad A \subseteq C}{(C - B) \to BD}(Unif)$$

The Unif rule can be seen as a special case of the Darwen's general unification theorem [9], and is easily checked to be sound, i.e. $\{A \to B, C \to D\} \models (C - B) \to BD$ provided that $A \subseteq C$ and $A \cap B = \emptyset$. By definition 3.2, we can say that $(C - B) \to BD$ implies $C \to D$, in other words, the FD $(C - B) \to BD$ is stronger than the FD $C \to D$. Therefore, the rule Unif computes stronger FDs.
The procedure Unif_rule aims to filter unnecessary FDs, that is dependencies that are not useful for computing keys. This will reduce the size of the dependencies set before computing the left closure. This is very important because the number of iterations for computing the closure is related to the size dependencies set. As a simple example of such a situation, consider the experiment #28 in table 1. Initially, we have $|\Gamma| = 2046$ and after calling the procedure of Unif_rule, the size of Γ is reduced to 11 dependencies, i.e, 2035 redundant dependencies are removed. This reduction will speed up the computation of attributes closure and then the keys. Transforming a redundant implicational system to a non-redundant one is a relevant topic in Formal Concept Analysis or FCA.

Definition 3.5. *The left closure of a set of FDs Γ, written Γ^*, is defined as $\Gamma^* = \{A \to A^* | A \to B \in \Gamma\}$.*

Theorem 3.1. *Let Γ^* be the left closure of Γ. If $A \to B \in \Gamma$, then $A \to B$ is implied by Γ^*.*

Proof. By definition 3.2, it is easy to see that for any FD $A \to B \in \Gamma$, there is an FD $C \to D \in \Gamma^*$ such that $C \to D$ *implies* $A \to B$. □

Lemma 3.1. *Let $R(\Omega, \Gamma)$ be a relation schema, Γ^* be the left closure of Γ and $A \to B \in \Gamma^*$. If $AB = \Omega$ (i.e. $A^+ = \Omega$), then A is a super key of R. The key A is minimal if every $a \in A$, we have $A - \{a\}$ is not a key.*

3.2 KeyFinder: the proposed algorithm

We present the proposed algorithm, *KeyFinder*, for discovering minimal keys of a relational schema. The general algorithm is given in algorithm 5. The main advantage here is the sub-routine *leftClosure* as it computes FDs with larger right-side. For a FD, larger the right-side more is the semantic information contained in that FD.

```
Algorithm 5: get_Keys
  Input: $\Omega$, $\Gamma$.
  Output: the List of all minimal keys $\mathcal{K}$ of $R(\Omega, \Gamma)$.
1 $\Omega' = body(\Omega, \Gamma)$ // use the rule closure to compute core$^+$, see the definition 3.1
2 if $\Omega' = \emptyset$ then
3     $\mathcal{K} = \{core(\Omega, \Gamma)\}$
4 else
5     $\Gamma' = \{X \cap \Omega' \to Y \cap \Omega' | X \to Y \in \Gamma\}$
6     $\Gamma" = Unif\_rule(\Gamma')$ // apply definition 3.4
7     $\Gamma^* = leftClosure(\Gamma")$ // apply definition 3.5
8     $\mathcal{K}' = Tableaux(\Omega', \Omega', \Gamma^*, \emptyset)$
9     $\mathcal{K} = \{k \cup core | k \in \mathcal{K}'\}$
```

```
Algorithm 6: Tableaux($\Omega$, $K$, $\Gamma$, $ACC$)
  Input: $\Omega$, the current attributes' set $K$, $\Gamma$, and the current list of super keys ACC
  Output: the list of all minimal keys $ACC$.
1  foreach $A \to B \in \Gamma$ do
2      if $(A \cup B) = \Omega$ then
3          $ACC = ACC \cup \{A\}$      // i.e. A is a key

4  foreach $A \to B \in \Gamma$ do
5      if not ImpliedFD($A \to B$, $ACC$) then
6
7          $K'$=$A(K - B)$      // create a direct child node K' of $K$;
8
9          $\Gamma' = \emptyset$      // $\Gamma'$ represents the new FD set
10         foreach $C \to D \in \Gamma$ do
11             $< X, Y >$=$sSimp(A \to B, C \to D)$
12             if $Y \neq \emptyset$ then
13                 $\Gamma' = \Gamma' \cup \{X \to Y\}$

14         if $\Gamma' = \emptyset$ then
15             $ACC = ACC \cup \{K'\}$      // K' is a key
16         else
17
18             Tableaux($\Omega$, $K'$, $\Gamma'$, $ACC$)
```

Example 3.3. *For our running example 2.2, the core $= \emptyset$, body $= \Omega$ and $\Gamma' = \Gamma$. The else-part of the algorithm 5 is then executed. Next, we remove possible redundant FDs and compute Γ^*, the left closure of Γ. For the FDs with lhs being a key, their lhs are directly added in the list of keys (lines 1-3, Algorithm 6), ACC, i.e., ACC={ab, bc, be, ce, cg}. This is justified by the lemma 3.1. Figure 6 illustrates the corresponding tableaux where we have only 5 nodes and 7 keys. Notice that there is no redundant key. As the root node is labeled with Γ^*, most of the keys are generated in one step. The reader can compare the two tableaux Figures 2 and 6. When using the parallel algorithm proposed in [3], we gets 11 nodes, 8 keys and one redundant key, see the Fig.3 pp 80.*

Example 3.4. *(Example 2.3 revisited) As in the previous example, we have core $= \emptyset$, body $= \Omega$ and $\Gamma' = \Gamma$. However, in this example we are facing the worst-case as $\Gamma^* = \Gamma$. When using our algorithm 5, we get the screenshot in Fig.7. We can see that the number of nodes is reduced from 19 to 12 with no duplicated keys. Therefore, our algorithm KeyFinder substantially improves the SST method.*

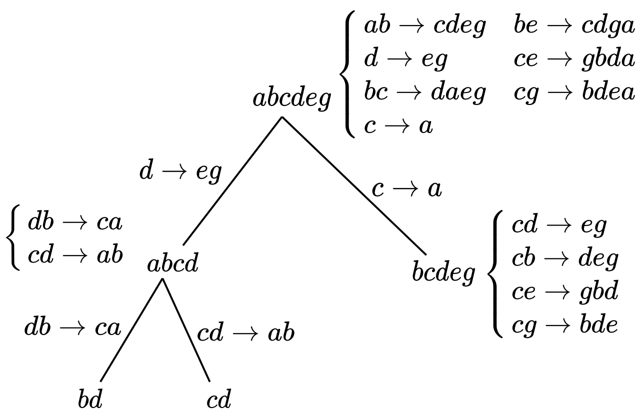
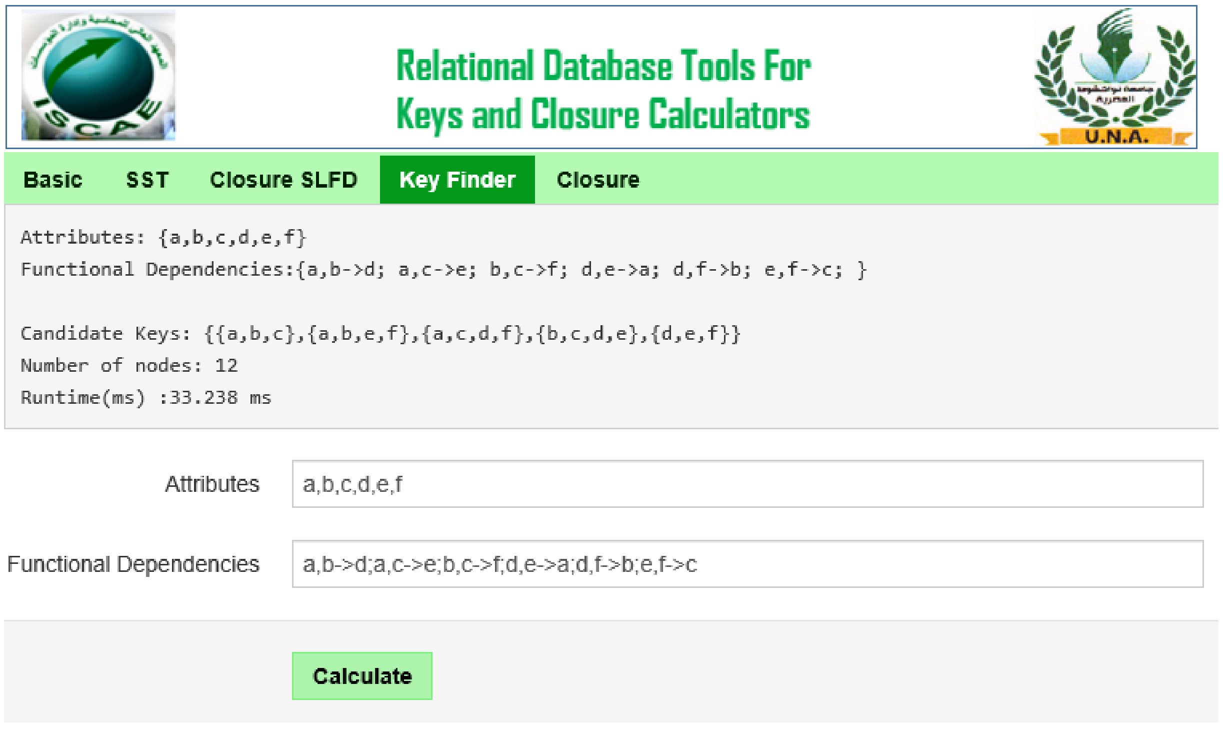


Figure 6: Illustration of *KeyFinder* on Example 2.2.



```
Attributes: {a,b,c,d,e,f}
Functional Dependencies:{a,b->d; a,c->e; b,c->f; d,e->a; d,f->b; e,f->c; }

Candidate Keys: {{a,b,c},{a,b,e,f},{a,c,d,f},{b,c,d,e},{d,e,f}}
Number of nodes: 12
Runtime(ms) :33.238 ms
```

Attributes	a,b,c,d,e,f
Functional Dependencies	a,b->d;a,c->e;b,c->f;d,e->a;d,f->b;e,f->c

Figure 7: Printout of KeyFinder on Example 2.3.

Proposition 3.1. *The rule of closure is sound.*

Proof. we have

1. $A \to B$ given
2. $C \to D$ given
3. $C \subseteq AB$ given
4. $A \to C$ by 1, 3 and the rule of *inclusion* (R1)
5. $A \to D$ by *transitivity* (R3) 4, 2

6. $A \to BD$ by *union* (R4) 1, 5

$\square$

Proposition 3.2. *(Soundness of Unif)*
Let Γ *be a set of functional dependencies. The rule of Unif is sound in the sense that all FDs which are derived with Unif are in the closure of* Γ.

Proof. One can use Armstrong's inference rules to show that the rule of *Unif* is sound:

1. $A \cap B = \emptyset$ given
2. $A \subseteq C$ given
3. $A \to B$ given
4. $C \to D$ given
5. from 2 and 3 and using the inclusion and transitivity rules, we have $C \to B$
6. from 4 and 5 and using the union rule, we have $C \to DB$
7. from 1, 2, we have $(C - B) \to DB$

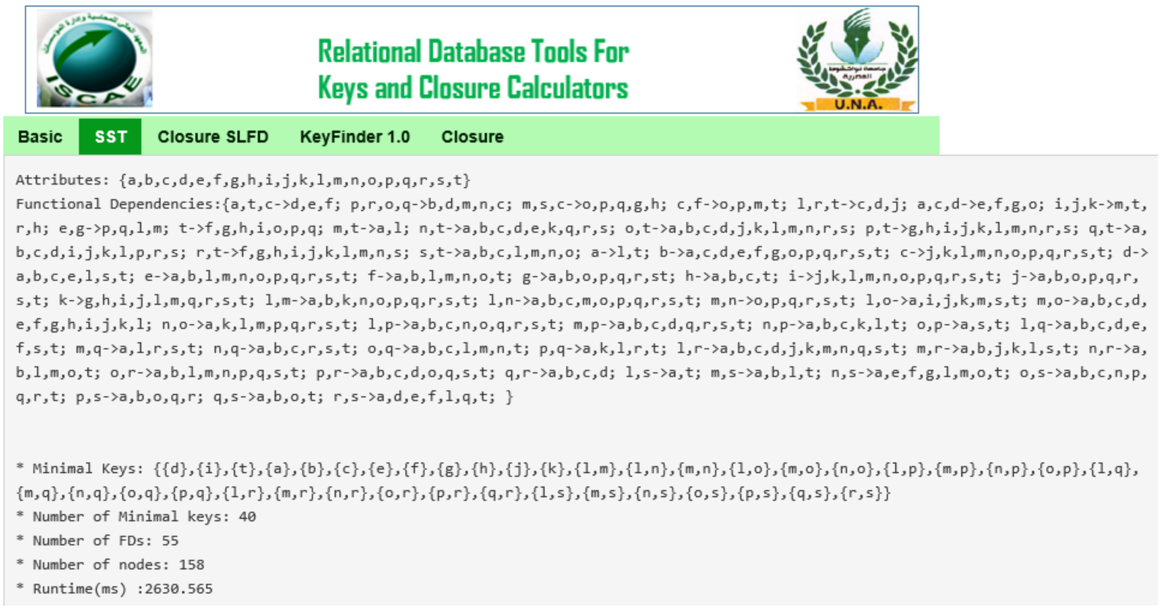
$\square$

4 Experimentation

In this work, we have proposed two methods: *KeyFinder* for computing minimal keys and *Closure* for computing attributes closure. All the algorithms presented here have been implemented in PHP. Two for the *key-finding problem*: *SST* and the proposed one *KeyFinder* and two for the *closure-finding problem*: SLD_{FD} and the proposed one *Closure*. We have also implemented the basic algorithm for finding keys. Although, the later is not tableaux-based.
Because of the lack of benchmarks for FDs, we have performed small-benchmark experiments in table 1 to show the effectiveness of the presented methods. Some of the examples are taken from the literature [23, 22, 27, 5, 8, 6, 10, 14] and others are randomly generated. The experiments #21 − #25 are taken from real-word data sets available at https://data.cms.gov/provider-data/[1][2]. Figures 8 and 9 show the outputs of the algorithms on example #23 that corresponds to *Unplanned hospital visits* data. In this experiment, *SST* generates 158 nodes while *KeyFinder* generates 1 node, the root node. Thus, *KeyFinder* calculates all the minimal keys before building the tableaux tree. In terms of execution time, the runtime of *KeyFinder* is almost a quarter of the runtime of *SST*.
In the other hand, the experimental evaluations are shown in table 1 where $|N_\alpha|$ is the number of nodes generated by the method α and $|T_\alpha|$ denotes the execution time of the method α.
Based on the experimental results, we can conclude that *KeyFinder* outperforms *SST* both in terms of search space (number of nodes) and execution time as illustrated in figures 10 and 11. The most time consuming part of *KeyFinder* is the *leftClosure* procedure. However, its introduction significantly improves both the search space and the execution time.
Experimental evaluations are conducted by considering two important statistical indicators: the search space and the runtime. The execution time is obtained by calculating the average value of several executions as it is closely related to machine architecture. We have omitted the parameter generated keys as both methods generate the same number of keys. In all conducted experiments, there is no case where our method had worse result than the *SST* method.
We run all our experiments on a Sony Intel(R) core(TM) i3-3110M CPU @ 2.40 GHz and 4.00 GB RAM. It is noteworthy that in our implementation the algorithm *SST* gets a significant improvement concerning the number of redundant keys. Notice that, the number of keys is not directly influenced by the number of attributes or FDs. We have used the t-test as a statistic method with $\alpha = 0.05$, we get the p-value=0.0026. This test confirms that *KeyFinder* is faster than *SST*.

[1]The data sets used are: Unplanned Hospital visits, Healthcare Associated Infections, Dialysis Adequacy Comprehensive measure, Outpatient and Ambulatory Surgery Consumer Assessment of Healthcare.

[2]For the sake of simplicity, we have renamed the attribute names with alphabetic letters a, b, c,...



```
Attributes: {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}
Functional Dependencies:{a,t,c->d,e,f; p,r,o,q->b,d,m,n,c; m,s,c->o,p,q,g,h; c,f->o,p,m,t; l,r,t->c,d,j; a,c,d->e,f,g,o; i,j,k->m,t,
r,h; e,g->p,q,l,m; t->f,g,h,i,o,p,q; m,t->a,l; n,t->a,b,c,d,e,k,q,r,s; o,t->a,b,c,d,j,k,l,m,n,r,s; p,t->g,h,i,j,k,l,m,n,r,s; q,t->a,
b,c,d,i,j,k,l,p,r,s; r,t->f,g,h,i,j,k,l,m,n,s; s,t->a,b,c,l,m,n,o; a->l,t; b->a,c,d,e,f,g,o,p,q,r,s,t; c->j,k,l,m,n,o,p,q,r,s,t; d->
a,b,c,e,l,s,t; e->a,b,l,m,n,o,p,q,r,s,t; f->a,b,l,m,n,o,t; g->a,b,o,p,q,r,st; h->a,b,c,t; i->j,k,l,m,n,o,p,q,r,s,t; j->a,b,o,p,q,r,
s,t; k->g,h,i,j,l,m,q,r,s,t; l,m->a,b,k,n,o,p,q,r,s,t; l,n->a,b,c,m,o,p,q,r,s,t; m,n->o,p,q,r,s,t; l,o->a,i,j,k,m,s,t; m,o->a,b,c,d,
e,f,g,h,i,j,k,l; n,o->a,k,l,m,p,q,r,s,t; l,p->a,b,c,n,o,q,r,s,t; m,p->a,b,c,d,q,r,s,t; n,p->a,b,c,k,l,t; o,p->a,s,t; l,q->a,b,c,d,e,
f,s,t; m,q->a,l,r,s,t; n,q->a,b,c,r,s,t; o,q->a,b,c,l,m,n,t; p,q->a,k,l,r,t; l,r->a,b,c,d,j,k,m,n,q,s,t; m,r->a,b,j,k,l,s,t; n,r->a,
b,l,m,o,t; o,r->a,b,l,m,n,p,q,s,t; p,r->a,b,c,d,o,q,s,t; q,r->a,b,c,d; l,s->a,t; m,s->a,b,l,t; n,s->a,e,f,g,l,m,o,t; o,s->a,b,c,n,p,
q,r,t; p,s->a,b,o,q,r; q,s->a,b,o,t; r,s->a,d,e,f,l,q,t; }


* Minimal Keys: {{d},{i},{t},{a},{b},{c},{e},{f},{g},{h},{j},{k},{l,m},{l,n},{m,n},{l,o},{m,o},{n,o},{l,p},{m,p},{n,p},{o,p},{l,q},
{m,q},{n,q},{o,q},{p,q},{l,r},{m,r},{n,r},{o,r},{p,r},{q,r},{l,s},{m,s},{n,s},{o,s},{p,s},{q,s},{r,s}}
* Number of Minimal keys: 40
* Number of FDs: 55
* Number of nodes: 158
* Runtime(ms) :2630.565
```

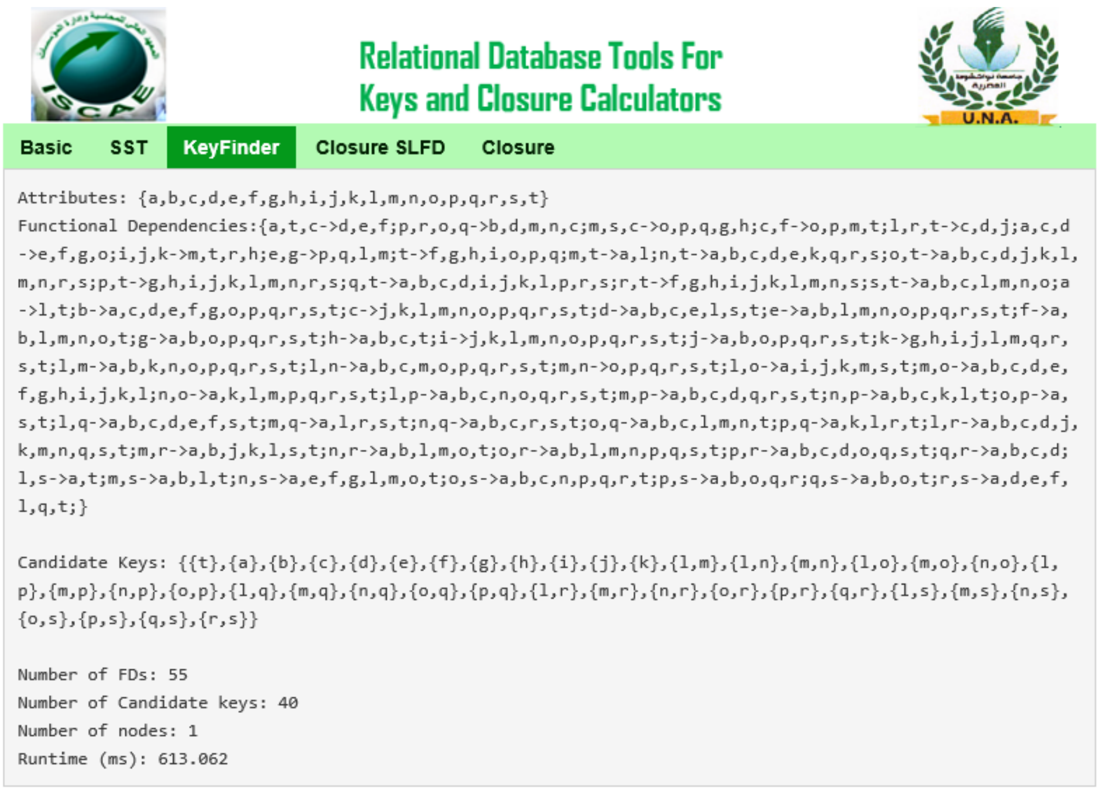
Figure 8: Screenshot of the SST method on Example #23.

5 Conclusion

A tableaux-based algorithm is presented in this paper. The new algorithm, *KeyFinder*, is able to derive all minimal keys of a relation schema. It uses a powerful derivation rule for computing the closure of a set of attributes. Experimental results show that *KeyFinder* improves drastically the *SST* method.
We have defined a strategy that consists to compute the left closure of FDs that allow us to reduce the search space. In contrast to the previous method that compute the closure of all subset of attributes, we only determine the closure of the relevant attributes .
We show that even in the worse case (when $body = \Omega$) indicating that there's no simplification of the initial problem in advance, *KeyFinder* is much better than *SST* in terms of search space.
The proposed algorithm *KeyFinder* can be improved by reducing the search space and then the execution time. Indeed, when computing the left closure of a given set of FDs, some FDs have already keys at their lhs. This information can be used to prevent the opening of new branches labeled with FDs whose lhs are keys. Figure 6 illustrates this case, all the FDs in Γ^* have keys at their lhs except the FDs $c \to a$ and $d \to eg$. It can also be applied to minimal generators in the terminology of formal concept analysis [13, 4].
As our future work, we plan to extract more FD sets from real-world data sets to further enrich our benchmark. We plan also to further improve the *leftClosure* procedure.

Acknowledgements

This research is funded by the National Agency of Scientific Research and Innovation.



```
Attributes: {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t}
Functional Dependencies:{a,t,c->d,e,f;p,r,o,q->b,d,m,n,c;m,s,c->o,p,q,g,h;c,f->o,p,m,t;l,r,t->c,d,j;a,c,d
->e,f,g,o;i,j,k->m,t,r,h;e,g->p,q,l,m;t->f,g,h,i,o,p,q;m,t->a,l;n,t->a,b,c,d,e,k,q,r,s;o,t->a,b,c,d,j,k,l,
m,n,r,s;p,t->g,h,i,j,k,l,m,n,r,s;q,t->a,b,c,d,i,j,k,l,p,r,s;r,t->f,g,h,i,j,k,l,m,n,s;s,t->a,b,c,l,m,n,o;a
->l,t;b->a,c,d,e,f,g,o,p,q,r,s,t;c->j,k,l,m,n,o,p,q,r,s,t;d->a,b,c,e,l,s,t;e->a,b,l,m,n,o,p,q,r,s,t;f->a,
b,l,m,n,o,t;g->a,b,o,p,q,r,s,t;h->a,b,c,t;i->j,k,l,m,n,o,p,q,r,s,t;j->a,b,o,p,q,r,s,t;k->g,h,i,j,l,m,q,r,
s,t;l,m->a,b,k,n,o,p,q,r,s,t;l,n->a,b,c,m,o,p,q,r,s,t;m,n->o,p,q,r,s,t;l,o->a,i,j,k,m,s,t;m,o->a,b,c,d,e,
f,g,h,i,j,k,l;n,o->a,k,l,m,p,q,r,s,t;l,p->a,b,c,n,o,q,r,s,t;m,p->a,b,c,d,q,r,s,t;n,p->a,b,c,k,l,t;o,p->a,
s,t;l,q->a,b,c,d,e,f,s,t;m,q->a,l,r,s,t;n,q->a,b,c,r,s,t;o,q->a,b,c,l,m,n,t;p,q->a,k,l,r,t;l,r->a,b,c,d,j,
k,m,n,q,s,t;m,r->a,b,j,k,l,s,t;n,r->a,b,l,m,o,t;o,r->a,b,l,m,n,p,q,s,t;p,r->a,b,c,d,o,q,s,t;q,r->a,b,c,d;
l,s->a,t;m,s->a,b,l,t;n,s->a,e,f,g,l,m,o,t;o,s->a,b,c,n,p,q,r,t;p,s->a,b,o,q,r;q,s->a,b,o,t;r,s->a,d,e,f,
l,q,t;}

Candidate Keys: {{t},{a},{b},{c},{d},{e},{f},{g},{h},{i},{j},{k},{l,m},{l,n},{m,n},{l,o},{m,o},{n,o},{l,
p},{m,p},{n,p},{o,p},{l,q},{m,q},{n,q},{o,q},{p,q},{l,r},{m,r},{n,r},{o,r},{p,r},{q,r},{l,s},{m,s},{n,s},
{o,s},{p,s},{q,s},{r,s}}

Number of FDs: 55
Number of Candidate keys: 40
Number of nodes: 1
Runtime (ms): 613.062
```

Figure 9: Screenshot of the KeyFinder method on Example #23.

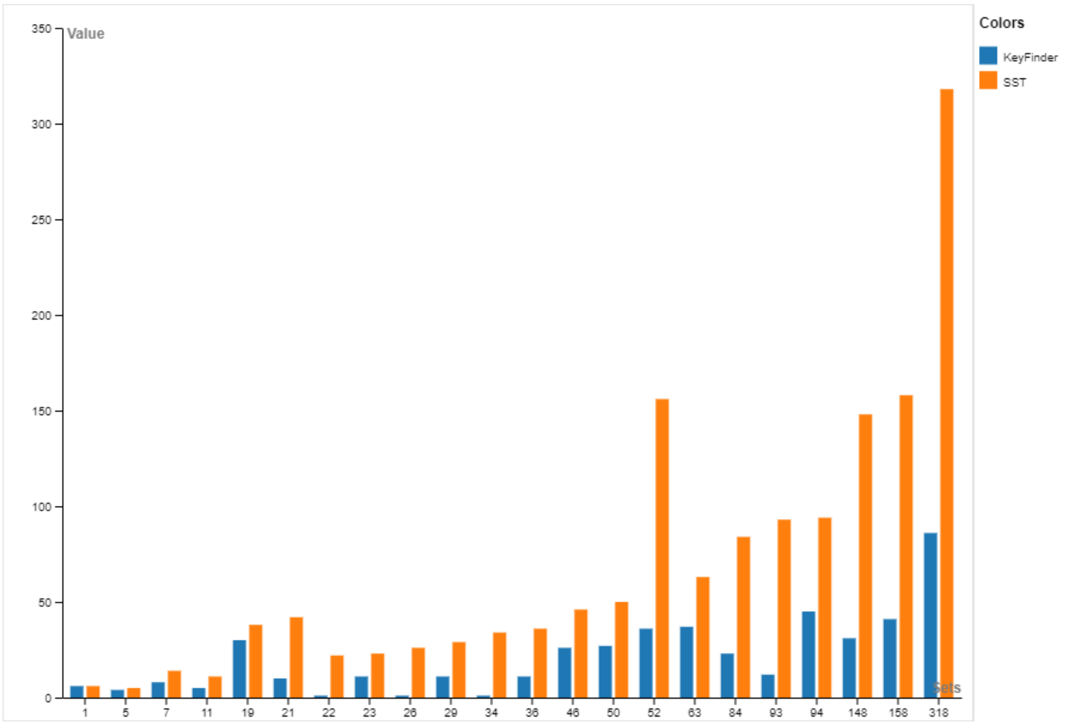


Figure 10: Comparing number of nodes.

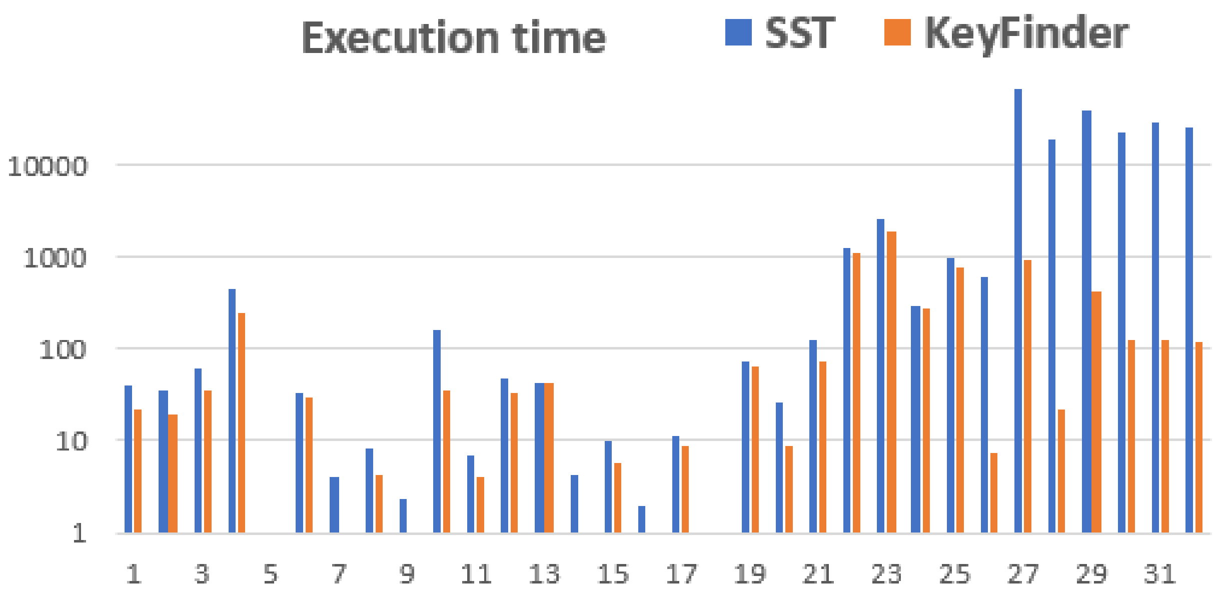


Figure 11: Comparing processing times

References

[1] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP congress*, volume 74, pages 580–583. Geneva, Switzerland, 1974.

[2] Amir Hassan Bahmani, Mahmoud Naghibzadeh, and Behnam Bahmani. Automatic database normalization and primary key generation. In *Canadian Conference on Electrical and Computer Engineering*, pages 000011–000016. IEEE, 2008.

[3] Fernando Benito-Picazo, Pablo Cordero, Manuel Enciso, and A Mora. Reducing the search space by closure and simplification paradigms. *The Journal of Supercomputing*, 73(1):75–87, 2017.

[4] Fernando Benito-Picazo, Pablo Cordero, Manuel Enciso, and Angel Mora. Minimal generators, an affordable approach by means of massive computation. *The Journal of Supercomputing*, 75(3):1350–1367, 2019.

[5] Subhrajyoti Bordoloi and Bichitra Kalita. Designing graph database models from existing relational databases. *International Journal of Computer Applications*, 74(1):219–231, 2013.

[6] Thomas M Connolly and Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[7] Pablo Cordero, Manuel Enciso, and Angel Mora. Automated reasoning to infer all minimal keys. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[8] Pablo Cordero, Manuel Enciso, Angel Mora, and I Pérez de Guzmán. A tableaux-like method to infer all minimal keys. *Logic Journal of the IGPL*, 22(6):1019–1044, 2014.

[9] Hugh Darwen and C Date. The role of functional dependencies in query decomposition. *Relational Database Writings*, 1991:133–154, 1989.

[10] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems*. Pearson, 2017.

[11] Raymond Fadous and John Forsyth. Finding candidate keys for relational data bases. In *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, SIGMOD'75, pages 203–210, New York, NY, USA, 1975. Association for Computing Machinery.

[12] M. Fernández and János Varga. Finding candidate keys and 3nf via strategic port graph rewriting. *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*, 2020.

[13] Bernhard Ganter and Rudolf Wille. *Formal concept analysis: mathematical foundations.* Springer Science & Business Media, 2012.

[14] Mark L Gillenson. *Fundamentals of database management systems.* John Wiley & Sons, 2008.

[15] JW Guan and David A Bell. Rough computational methods for information systems. *Artificial intelligence*, 105(1-2):77–103, 1998.

[16] Radek Janostik, Jan Konecny, and Petr Krajča. Lincbo: fast algorithm for computation of the duquenne-guigues basis. *Information Sciences*, 572:223–240, 2021.

[17] Sukhamay Kundu. An improved algorithm for finding a key of a relation. In *Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 189–192, 1985.

[18] Claudio L Lucchesi and Sylvia L Osborn. Candidate keys for relations. *Journal of Computer and System Sciences*, 17(2):270–279, 1978.

[19] David Maier. Minimum covers in relational database model. *Journal of the ACM (JACM)*, 27(4):664–674, 1980.

[20] Angel Mora, Gabriel Aguilera, Manuel Enciso, Pablo Cordero, and Inmaculada Perez de Guzmán. A new closure algorithm based in logic: Slfd-closure versus classical closures. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 10(31):31–40, 2006.

[21] Angel Mora, Pablo Cordero, Manuel Enciso, Inmaculada Fortes, and Gabriel Aguilera. Closure via functional dependence simplification. *International Journal of Computer Mathematics*, 89(4):510–526, 2012.

[22] E Pichat. Algorithme de décomposition de clés. *RAIRO. Informatique théorique*, 19(3):213–232, 1985.

[23] Hossein Saiedian and Thomas Spencer. An efficient algorithm to compute the candidate keys of a relational database schema. *The Computer Journal*, 39(2):124–132, 1996.

[24] Jeffrey D Ullman. *Principles of database systems.* Galgotia publications, 1984.

[25] Jeffrey D Ullman. Principles of database and knowledge-base systems. *Computer Science Press*, 1989.

[26] Ralf Wastl. Linear derivations for keys of a database relation schema. *Journal of Universal Computer Science*, 4(11):883–897, 1998.

[27] Peter B Worland. An efficient algorithm for 3nf determination. *Information Sciences*, 167(1-4):177–192, 2004.

Table 1: Random benchmark.

#	$\|\Gamma\|$	N_{SST}	$N_{KeyFinder}$	difference	T_{SST} (ms)	$T_{KeyFinder}$ (ms)	difference
1	6	21	5	16	40.814	22.067	18.747
2	6	19	12	7	34.527	19.238	15.289
3	7	29	11	18	61.051	35.152	5.899
4	10	318	86	232	436.12	248.12	187.997
5	5	1	1	0	0.03	0.03	0
6	7	23	11	12	32.646	28.86	3.786
7	7	1	1	0	4.02	0.03	3.99
8	6	5	4	1	8.346	4.371	3.971
9	4	1	1	0	2.386	0.03	2.356
10	7	63	37	26	160	35	125
11	10	7	4	3	6.799	3.94	2.859
12	7	36	11	25	48.128	33.359	14.769
13	7	46	26	20	42.573	42.364	0.209
14	8	1	1	0	4.234	0.034	4.2
15	10	7	4	3	9.95	5.897	4.053
16	3	1	1	0	1.95	0	1.95
17	8	11	5	6	11.547	8.625	2.922
18	7	1	1	0	0.032	0.032	0
19	10	50	27	23	75.167	65.815	9.352
20	8	21	5	16	25.856	8.89	16.966
21	13	26	1	25	123.025	74.372	48.653
22	23	94	45	49	121.278	1112.2	109.055
23	55	158	1	157	2630.565	613.062	2017.503
24	17	19	18	1	301.068	278.81	2.257
25	30	84	23	61	993.08	793.11	199.97
26	126	22	1	21	592.507	7.433	585.074
27	1870	148	31	117	68713.082	938.98	67774.1
28	2046	34	1	33	19327.512	21.523	19305.989
29	1987	93	12	81	39362.808	417.46	38945.345
30	2019	52	12	40	22851.253	128.739	22722.514
31	2003	52	12	40	28634.065	127.186	28506.879
32	1955	52	12	40	25687.111	118.663	25568.448
33	1904	45	8	37	23200.643	107.654	23092.989
34	1876	74	23	51	27568.102	137.453	27430.649
35	1789	85	26	59	31763.107	139.364	31749.743