J Supercomput (2017) 73:75–87
DOI 10.1007/s11227-016-1622-1

Reducing the search space by closure and simplification paradigms

A parallel key finding method

F. Benito-Picazo[1] · P. Cordero[1] · M. Enciso[1] · A. Mora[1]

Published online: 14 January 2016

Abstract In this paper, we present an innovative method to solve the minimal keys problem strongly based on the Simplification Logic for Functional Dependencies. This novel method improves previous logic-based methods by reducing, in a significant degree, the size of the search space this problem deals with. Furthermore, the new method has been designed to easily fit within a parallel implementation, thereby increasing the boundaries current methods can reach.

Keywords Minimal keys · Parallelism · Logic

1 Introduction

Finding a set of items/attributes characterizing a greater set of data is a significant problem in several areas: data modeling, data integration, integration of heterogeneous databases, knowledge discovery, anomaly detection, query formulation, query optimization, and indexing. This problem is also interesting in emergent areas as

✉ F. Benito-Picazo
fbenito@lcc.uma.es

P. Cordero
pcordero@uma.es

M. Enciso
enciso@uma.es

A. Mora
amora@ctima.uma.es

[1] University of Málaga, Andalucía Tech, Málaga, Spain

linked data. In [14] authors delimit the problem: "establishing semantic links between data items can be really useful, since it allows crawlers, browsers and applications to combine information from different sources". The proposal is the use of keys to infer identity links among the data distributed in the web.

The notion of key is fundamental in Codd's relational model of data [2]. Calculating all minimal keys constitutes a tough problem; in [10,22], authors prove that the number of minimal keys for a relational system can be exponential in the number of attributes, or factorial in the number of dependencies. The main approaches to this problem point to the works of Lucchesi and Osborn in [10] that show an algorithm to calculate all candidate keys. Saiedian and Spencer [15] presented an algorithm using attribute graphs to find all possible keys of a relational database schema, and Wastl [20] proposed an inference system for deriving all keys which can be inferred from a set of functional dependencies. Recent works about calculating minimal keys are [16,21] and a very modern paper approaching the problem in a logic style [5]. In addition, in [9,18,19] the authors proposed the use of formal concept analysis [6] to face problems related with the discovery and management of implications, which can be considered complementary to our work.

We highlight [20] to be the one closest to our framework. In that work, a tableaux method to calculate all minimal keys is presented. In [3] we introduce the Key Algorithm, a method inspired in Wastl's tableaux method where the Simplification Logic for Functional Dependencies [12] is used to find all minimal keys. Later, in [5] we introduce the SST method, achieving a great performance in comparison to its predecessors based on the introduction of the inclusion-minimality test which avoids the opening of extra-branches of the tree, and the search space becomes narrower.

A parallel algorithm for the manipulation of implications is described in [17], enclosed in the field of hyper-graphs. Parallel issues in similar problems have been addressed by several authors. For instance, Krajca et al. [8] present a parallel algorithm for computing formal concepts. A first view for the parallelization of Wastl's method and the Key Algorithm was presented in [1], showing how parallelism could naturally be integrated in tableaux methods.

This paper makes the following contributions. We propose a new method named Closure Keys incorporating an efficient prune mechanism that use the closure method based on $\mathbf{SL}_{\mathrm{FD}}$ to improve SST method. The new method is based on the strong relation between the notion of key and the closure operator, not only in the definition issue but also in its construction. The closure operator defined in [12] allows us to highly reduce the search space by introducing shortcuts in the way to the leaves, where keys are finally produced. Moreover, a parallel implementation of SST method is presented along with several experiments to confirm the improvements that have been accomplished.

The remainder of the paper is organized as follows: Sect. 2 presents the key finding problem and the state of the art. In Sect. 3, we explain how a closure operator could help us improving the efficiency over minimal keys methods. In Sect. 4, the main aspects of the implementation of these methods will be exposed. Several results are shown in Sect. 5 with a collection of tables and charts. Finally, conclusions will end up the document.

2 Background

In this section we introduce the minimal key problem, the basic notions related with this issue and the last logic-based algorithm in the literature to tackle this problem.

Definition 1 (*Functional dependency*) Let Ω be a set of attributes. A functional dependency (FD) over Ω is an expression of the form $X \rightarrow Y$, where $X, Y \subseteq \Omega$. It is satisfied in a table R if for every two tuples of R, if they agree on X, then they agree on Y.

A key of a relational table is an attribute subset that allows us to uniquely characterize each row, and it is defined by means of FDs as follows:

Definition 2 (*Key*) Given a table R over the set of attributes Ω, we say that K is a key in R if the functional dependency $K \rightarrow \Omega$ holds in R.

We would like to manage keys with no superfluous attributes, named minimal keys. Due to space limitation, we refer those readers non-familiar with the formal notions of FDs, keys and relational tables to [11]. In addition, it is remarkable that this classical problem appears in several areas. For instance, in [5], we show how the minimal key problem in databases has an analogous one in formal concept analysis, where the role of FDs is played by attribute implications. In that paper, minimal key problem was presented from a logical point of view. An axiomatic system to manage both FDs and implications, named Simplification Logic and denoted $\mathbf{SL}_{\mathrm{FD}}$, was introduced in [4]. The inference system of such logic is presented as follows:

Definition 3 (*Axiomatic system*) $\mathbf{SL}_{\mathrm{FD}}$ has a reflexivity axiom `[Ref]` $\frac{B \subseteq A}{A \rightarrow B}$; and inference rules named fragmentation, composition and simplification.

$$\texttt{[Frag]}\ \frac{A \rightarrow BC}{A \rightarrow B}; \qquad \texttt{[Comp]}\ \frac{A \rightarrow B,\ C \rightarrow D}{AC \rightarrow BD}; \qquad \texttt{[Simp]}\ \text{If } A \subseteq C, A \cap B = \varnothing,\ \frac{A \rightarrow B,\ C \rightarrow D}{C\text{-}B \rightarrow D\text{-}B}$$

This logic allows the development of efficient methods to manage FDs and implications. In [5] a new algorithm, named SST, for computing all minimal keys using a tableaux-like strategy was introduced, opening the door to embed a massive parallelism into its execution.

SST relies on the notion of set closure, a basic notion in database theory which allows to characterize the maximum attribute set that can be reached from a given attribute set A with respect to a set of FDs using the axiomatic system. Thus, if the closure of A is denoted as A_Γ^+, the inference system for FDs allows us to infer the FD $A \rightarrow A_\Gamma^+$. The logical-style approach to the minimal key problem consists in the enumeration of all attribute sets A such that the following FD holds $A \rightarrow \Omega$.

SST algorithm is strongly based on two proper extensions of the $\mathbf{SL}_{\mathrm{FD}}$ simplification rule, named `[sSimp]` and `[lSimp]`. These rules guide the construction of the search space to look for all minimal keys (see [5] for further details). The method works step by step by building a tree from the original problem to the solution, i.e. the set of minimal keys. They are applied to each pair (attribute set, implication set) labeling each node to produce new subnodes that will be the origin of new branches in the search space. Applying `[lSimp]` to the subset of attributes in each node and the implication

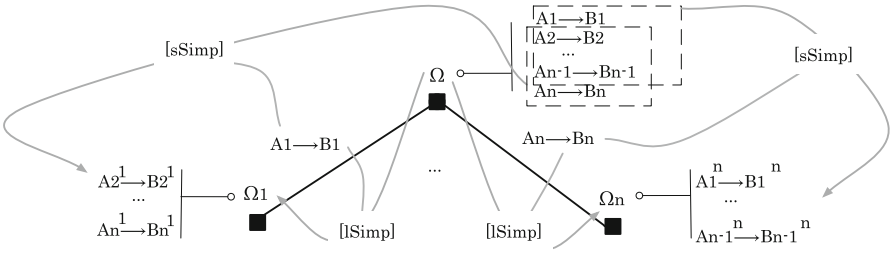

Fig. 1 `[lSimp]` generates new implication subsets and `[lSimp]` generates new roots

in the corresponding edge, we obtain the new root in the branch (e.g. in Fig. 1 from Ω and $A_1 \rightarrow B_1$, we obtain the new subset Ω_1). Moreover, the application of `[sSimp]` over the implication set in each node is done by taking every implication as a pivot and applying the rule to the rest of the implications with the corresponding pivot to generate new branches (e.g. in Fig. 1 the first branch is generated by taking $A_1 \rightarrow B_1$ as the pivot and facing it to the rest of the implication set $A_2 \rightarrow B_2, \ldots, A_n \rightarrow B_n$).

When the set of implications in a node is the empty set, we have reached a leaf of the tree and a minimal key is added to the solution. The full situation is depicted in Fig. 1, where a schema of the application of both rules to a generic node is shown.

SST shows a great performance compared with its predecessors. The main benefit in the reduction of the search space was the introduction of the inclusion-minimality test to avoid the opening of extra-branches. Thus, SST does not open some branches which are going to produce the same keys that are calculated in another branch. The characterization of such branches rendering leaves with duplicate information is not a trivial issue. To approach it, we have defined the notion of minimal implication w.r.t. an implication set Γ: $A \rightarrow B \in \Gamma$ is minimal if for all $C \rightarrow D \in \Gamma$ we have that $C \not\subset A$.

3 Improving minimal keys methods by means of closure operator

As it has been established this far, the enumeration of all minimal keys is a non-trivial problem that can be approached using logic-based methods. The main goal to progress in this line is the reduction of the search space that can be measured by means of the number of nodes in the tree. In the previous section, we have summarized the main characteristics of the SST method and how it incorporates a strategy to shorten the width of the tree corresponding to the search space. Thus, it provides a narrower tree regarding previous methods. In this spirit, we have studied how to reduce even more the size of the tree by shorting its depth. The kernel of the method presented here is a logic-based closure algorithm presented in [12], strongly based on Simplification Logic $\mathbf{SL}_{\mathrm{FD}}$. The closure of an attribute set can be solved in linear time w.r.t. the cardinality of the implication set. There exists in the literature several approaches to tackle this problem, but most of them may be considered a modification of classical Maier's method [11].

The method presented in [12], named $\mathbf{SL}_{\mathrm{FD}}$ Closure, can be considered a novel approach to classical closure methods, because it provides a new feature: In addition to the set of attributes that constitutes the output of the closure operator, $\mathbf{SL}_{\mathrm{FD}}$Closure provides a subset of implications of the original Γ set. This new implication subset encloses the knowledge that can be considered as the complement of the closure and, as we shall see, may be used in a very smart way to find minimal keys by means of the closure operator.

As mentioned before, closure operator may be used to define the notion of key. In this section we propose it as the backbone of a new method to compute all minimal keys. In this approach, $\mathbf{SL}_{\mathrm{FD}}$ Closure plays an outstanding role. The main point of our method is that it receives a set of implications Γ and a subset of attributes $X \subseteq \Omega$. It renders the closure set X^+ w.r.t. Γ, and also, a novelty output Γ' consisting in a new implication set that covers the semantics that reaches beyond X^+. If $\Gamma' = \varnothing$ then $X^+ = \Omega$ (see [12] for further details).

We propose to apply $\mathbf{SL}_{\mathrm{FD}}$ Closure to each minimal implication in the Γ set in each node to open new branches with the output produced by $\mathbf{SL}_{\mathrm{FD}}$ Closure. The definition of our new minimal key method is presented in Algorithm 1. An illustrative example showing the reduction in the tree provided by Closure Keys w.r.t. SST method is shown in Figs. 2 and 3 respectively.

The main call of Algorithm 1 must be Closure_Keys(Ω, Γ, $\varnothing$, $\varnothing$). Later, to get all minimal keys we pick up the minimal elements of the fourth parameter (which acts as an accumulator in a in/out mode) of this procedure call with regard to $(2^{\Omega}, \subseteq)$. The following theorem ensures that Algorithm 1 provides a sound and complete method.

Theorem 1 *Let Ω be an attribute set and Γ be an implicational system over M. Then Algorithm 1 renders all minimal keys.*

Proof Firstly, the termination of the algorithm is ensured because, in each recursive call of Closure_Keys, the cardinality of the implicational system is strictly decreased.

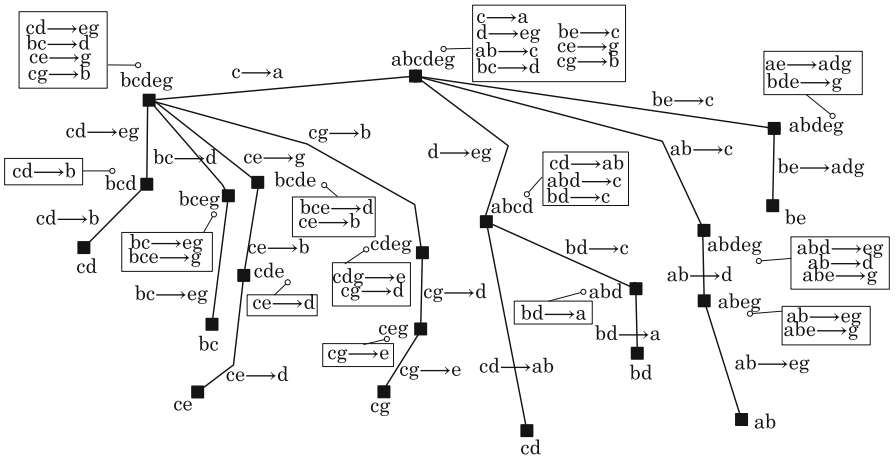

Fig. 2 A complete example using SST method

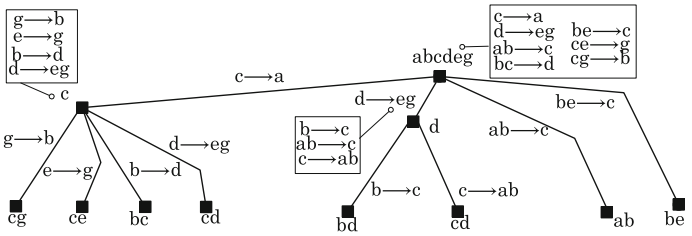

Fig. 3 $\mathbf{SL}_{\mathrm{FD}}$ Closure Keys Method example. Notice how the treatment of the implication set in each node produces a big reduction of the tree of Fig. 2 from 21 to 11 nodes

Moreover, the number of recursive calls is limited by the cardinality of Γ. The kernel of the method is the $\mathbf{SL}_{\mathrm{FD}}$ Closure presented in [12] providing a compound output, i.e., $\mathbf{SL}_{\mathrm{FD}}\mathit{Closure}(X, \Gamma) = \langle X^+, \Gamma' \rangle$.

This method produces all minimal keys because, in each step, it renders a closure X^+. If it is not the entire Ω, we go ahead by selecting a new antecedent from the implication set Γ'. We remark that this implication set has been depurated by means of $\mathbf{SL}_{\mathrm{FD}}$Closure. By virtue of this, once we reached a leaf, the union of the antecedents in the way through conform a key and, therefore, the method is sound.

Concerning its completeness, any key in the system is produced since the search space induced by our method constitutes an exhaustive search. As a conclusion from the key study presented in [13], keys have to be composed with bricks being the antecedent of the implications. Those pruned branches do not fall down the completeness since the removed keys were redundant.

Algorithm 1: Closure_Keys

Data: Ω a set of attributes, Γ a set of implications, $\mathcal{C}$ an accumulator variable of a subset set of attributes, $\mathcal{K}$ a acumulator variable with sets of attributes.

```
begin
    A:=Ω
    if Γ = ∅ then Add(K, {A ∪ C})
    else
    foreach X→Y ∈ [IM](Γ) do
        < X', Γ' >:=SL_FD Closure(X, Γ)
        C := C ∪ X
        Closure_Keys(X', Γ', C, K)
    /* Where [IM](Γ) denotes the implications in Γ being minimal FD. */
end
```

4 Implementation

The approach presented determines that, due to the construction of the search tree, parallel strategies and big hardware resources are strongly needed when we try to resolve problems containing substantial size at the input. In order to test the improvement, we are going to proceed with a parallel implementation of SST and Closure_Keys algorithms that works in two steps, (1) a partial method splits the entry problem into several subproblems and (2) a parallel method resolves all these generated subproblems and composes the final result, the minimal key set (see Fig. 4).

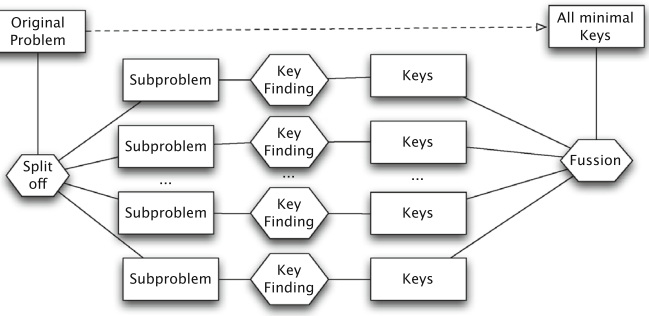

Fig. 4 Parallelism diagram

We can go along this kind of strategy over our parallel implementations because, due to the tableaux nature, each branch will be totally independent to others, so every node can be treated on its own. That is the great point of using parallelism within these techniques; we can deliver a huge number of problems to different cores so they can be solved simultaneously. By virtue of this, the size of the problems at the input could be greater without exceeding the machine limitations; at least, we have distanced the limit in a substantial way for the moment.

The application of the partial code moves us to split the entry problem into several subproblems. At this regard, we will stop at a certain level of the tree, and there, every node constitutes an independent problem by itself. Each of these problems represents a branch of the search tree and will be the input for the application of the parallel method, that will end up in a leaf of the tree. However, it is needed to go forward determining the value that will decide the splitting point, and this is, indeed, not an easy issue so far, as we discuss in the following paragraph.

As break-off value (BOV), determining the level in which the branch is considered an atomic to be treated by the parallel stage, we use the cardinal of the set of implications of the current node, since we have observed that, the bigger this cardinal is, the longer the branch *used to* reach. Once we have all these subproblems, and due to all of them conform an individual state of the algorithm, they are meant to be solved by the parallel code. However, if we are treating with big problems (those containing a substantial number of attributes and implications), the number of generated subproblems could be huge (we have run experiments with 50,000+ subproblems). Thus, handling them is a task only available to a significant amount of resources, as Supercomputing and Bioinnovation Center of the University of Málaga[1] supports us.

5 Experiments and results

We have randomly generated a set of experiments, varying the number of attributes and implications. We start with problems with 100 implications, each one built using 100 different attributes as well. Then, we move forward considering bigger problems

[1] http://www.scbi.uma.es.

Table 1 Attempts trying to improve execution times by increasing the number of cores

Problem and method	Implications 150 Partial$_t$ (s)	Attrib 150 Total$_t$ (s)	BOV 140 Nodes	Cores	Ratio
150150-4-SST	581	885	55.211	32	62
150150-4-CK	48	65	25.477	32	391
150150-4-SST	576	880	55.211	64	62
150150-4-CK	46	64	25.477	64	398

counting on 150 implications and 150 attributes. Notice that these numbers go far beyond machine capabilities, as it has been demonstrated in previous studies of this work [3]; they even substantially exceed the results given in [1], where parallelism techniques had already been applied.

Obviously, both methods obtain the same keys. Therefore, we have omitted this parameter in result tables since it only experimentally validates the method. Consequently, in order to compare SST method versus Closure Keys method, we focus on execution times and the number of nodes of the tree (so, we can figure out the size of the problem).

Number of nodes will always be the same for an individual experiment no matter the number of runs we do. However, execution times will not go along exactly with this statement. They could be slightly different due to it intrinsic nature. Differences concerning number of nodes will show how new methods have improved the algorithm as the depth of the tree has been sharply reduced, and consequently, execution times become better. Besides, we have included a last column in order to show the ratio of time and nodes between SST and CK. Regarding all these points, needless to say that every execution time shown here is the fruit of a post-statistical study [7,23] behind the results obtained from every run, so we kept the most reliable ones.

Before we reach the results section, the hardware architecture that has been used for developing every test shown in the paper can be visited in http://www.scbi.uma.es/site/scbi/hardware.

Every experiment we show here have been carried out using a cluster of 32 cores from those available within the hardware architecture mentioned above. Few subsequent experiments increasing the number of cores available went not up to expectations results in terms of execution times, as we briefly show in Table 1. Increasing the number of cores to engage parallelism within these kind of problems is a matter where much still remains to be investigated.

The first experiment starts solving a battery of hard load problems counting on 100 attributes and 100 implications. Results are shown in Table 2.

Even with such a big number of attributes and dependencies, it's been just around 10 min long for the slowest algorithm to finish (problem 100100-3). Yet, less than 300k nodes have been the maximum size of the Tableaux of all of them (problem 100100-7). It is absolutely out of mind thinking about solving these problems using sequential versions of the algorithms, since execution times would go out of hands.

Table 2 Parallel methods applied to big-size problems (I)

Problem and method	Attrib 100 Subp	Implications 100 Partial$_t$ (s)	BOV 90 Total$_t$ (s)	Cores 32 Nodes	Ratio
100100-1-*SST*	14	0	1	33	33
100100-1-*CK*	0	0	0	15	15
100100-2-*SST*	1.354	36	105	25.621	244
100100-2-*CK*	212	4	15	12.715	847
100100-3-*SST*	8.602	183	644	192.574	299
100100-3-*CK*	1.286	37	99	94.255	952
100100-4-*SST*	400	7	26	1.704	65
100100-4-*CK*	15	1	2	751	375
100100-5-*SST*	39	0	2	119	59
100100-5-*CK*	0	0	1	42	42
100100-6-*SST*	1.808	37	123	7.856	63
100100-6-*CK*	115	4	9	3.698	410
100100-7-*SST*	6.167	182	489	275.429	563
100100-7-*CK*	1.378	24	90	118.884	1.320
100100-8-*SST*	5.104	146	415	182.167	438
100100-8-*CK*	1.014	19	68	81.632	1.200
100100-9-*SST*	314	11	25	868	34
100100-9-*CK*	0	1	1	341	341
100100-10-*SST*	1.130	27	84	12.541	149
100100-10-*CK*	136	4	10	6.128	612

In fact, these are encouraging results since, earlier experiments developed using older methods [1] couldn't even have imagined to deal with such a huge problems.

This first experiment confirms which it was mentioned above: subproblems and nodes show a better performance by Closure Keys method.

A very remarkable outcome can be appreciated in problems 100100-{1,5,9}. Notice that Closure Keys does not create any subproblems. As this regard, the improvement is twofold: (1) in some cases, with the same BOV, the new method doesn't even need to move on the parallel implementation, partial one is enough to resolve these problems, and (2) to exploit this discovery, at the moment we need to deal with even more complex problems, we can establish a BOV nearer to the root of the tree and then, partial time will be significantly reduced.

Second experiment goes ahead giving another turn of the screw increasing the input of our experiments. We develop a new battery of 10 problems, increasing the number of implications and available attributes up to 150.

As far as these problems become more complex, the better is the improvement achieved by the new method. Executions times and number of nodes have been drastically reduced as shown in Table 3. Actually, we decided to add one more experiment (problem 150150-EXTRA) due to the specific remarkable results it reached.

Table 3 Parallel methods applied to big-size problems (II)

Problem and method	Attrib 150 Subp	Implications 150 Partial$_t$ (s)	BOV 140 Total$_t$ (s)	Cores 32 Nodes	Ratio
150150-1-*SST*	165	6	14	911	65
150150-1-*CK*	11	2	3	374	124
150150-2-*SST*	2.949	229	394	116.517	295
150150-2-*CK*	347	25	44	54.375	1.235
150150-3-*SST*	12.968	1.049	1.716	157.947	92
150150-3-*CK*	822	125	165	68.531	415
150150-4-*SST*	5.352	581	885	55.211	62
150150-4-*CK*	344	48	65	25.477	391
150150-5-*SST*	5.361	211	484	32.377	66
150150-5-*CK*	168	27	36	12.522	347
150150-6-*SST*	771	72	155	17.298	111
150150-6-*CK*	79	7	11	8.110	737
150150-7-*SST*	9.473	638	1.252	576.912	460
150150-7-*CK*	1.754	97	187	262.621	1.404
150150-8-*SST*	5.466	424	857	510.627	595
150150-8-*CK*	966	57	104	257.267	2.473
150150-9-*SST*	235	25	45	3.632	80
150150-9-*CK*	24	3	4	1.726	431
150150-10-*SST*	3.403	348	555	102.537	184
150150-10-*CK*	277	31	46	45.962	999
150150-EXTRA-*SST*	31.401	2.950	30.983	21,404,732	690
150150-EXTRA-*CK*	8.049	354	1.320	10,614,386	8.041

Several experiments are worth to be discussed separately. As a matter of fact, if we set our sight on problems 150150-{3,7}, number of subproblems generated are much less for Closure Keys method than SST. Difference is not trivial so far, it also guides us directly to a great reduction of partial execution times. Total execution times go along in the same direction. Regarding the size of the tree, the benefits by introducing the closure are pretty striking. Most of cases, the number of nodes becomes near to 50 % lower. Figure 5 would help us appreciating the differences more clearly.

6 Conclusions

We presented here a method to solve the key finding problem. Among the different strategies introduced in the literature, we follow the logic-based line pioneered in [20]. The new method presented here uses the Simplification Logic and introduces an alternative to the SST method presented in [5]. Here, we provide a further reduction of the tree size by shortening its depth, empirically confirmed by means of the number of

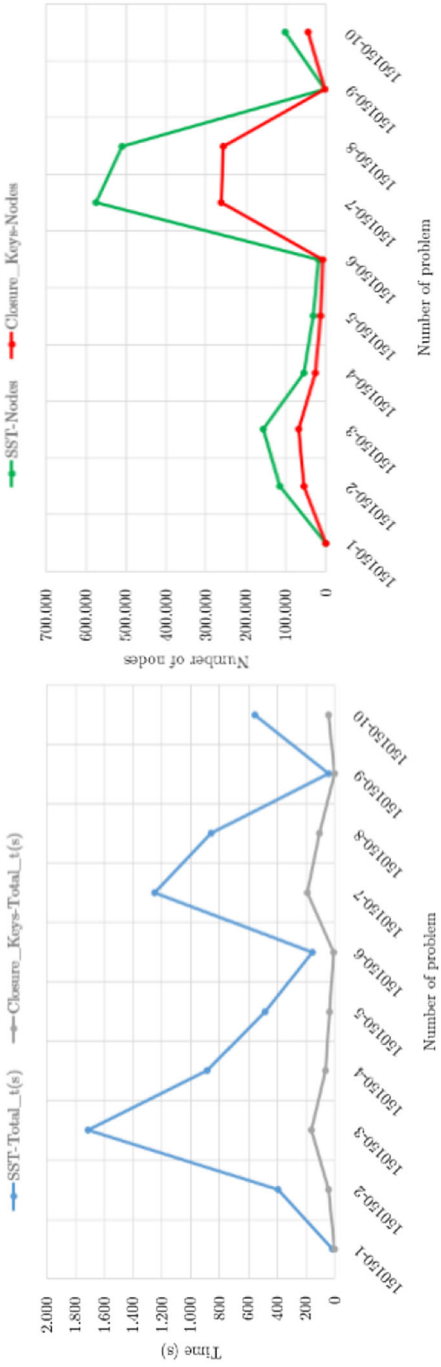

Fig. 5 Execution times and number of nodes for Big-size problems (II)

generated nodes. In addition, a parallel implementation has been developed following a map-reduce architecture. The inherent parallel design of the new method allows us to face problems containing a significant size at the input.

We have set two priorities for future works:

- Applying these mechanisms over real data stores, so we can figure out how they really could take advantage of parallelism working on these kind of datasets.
- The study of the impact concerning the number of cores involved in the experiments.

Acknowledgements Supported by Grants TIN2011-28084 and TIN2014-59471-P of the Science and Innovation Ministry of Spain, co-funded by the European Regional Development Fund (ERDF). The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the SCBI (Supercomputing and Bioinnovation) center of the University of Málaga (Andalucía Tech).

References

1. Benito-Picazo F, Cordero P, Enciso M, Mora A (2014) Increasing the efficiency of minimal key enumeration methods by means of parallelism. In: ICSOFT-EA 2014—proceedings of the 9th international conference on software engineering and applications, Vienna, Austria, 29–31 August 2014, pp 512–517
2. Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387
3. Cordero P, Enciso M, Mora A (2013) Automated reasoning to infer all minimal keys. In: Proceedings of the twenty-third international joint conference on artificial intelligence, IJCAI'13. AAAI Press, pp 817–823
4. Cordero P, Enciso M, Mora A, de Guzmán IP (2002) $\mathrm{Sl}_{\mathrm{fd}}$ logic: elimination of data redundancy in knowledge representation. In: Advances in artificial intelligence 8th Ibero-American conference on AI, Seville, Spain, November 12–15, pp 141–150
5. Cordero P, Enciso M, Mora A, de Guzmán IP (2014) A tableaux-like method to infer all minimal keys. Log J IGPL 22(6):1019–1044
6. Ganter B, Wille R (1997) Formal concept analysis: mathematical foundations, 1st edn. Springer-Verlag New York Inc, Secaucus
7. Goh TN (2010) An information management paradigm for statistical experiments. Qual Reliab Eng Int 26(5):487–494
8. Krajca P, Outrata J, Vychodil V (2008) Parallel recursive algorithm for FCA. In: 6th International conference on concept lattices and their applications (CLA), vol 433. CEUR WS, Olomouc, pp 71–82 (2008)
9. Lèvy G, Baklouti F (2005) A distributed version version of the ganter algorithm for general galois lattices. In: 3rd International conference on concept lattices and their applications (CLA), vol 162. CEUR WS, Olomouc, pp 207–221
10. Lucchesi CL, Osborn SL (1978) Candidate keys for relations. J Comput Syst Sci 17(2):270–279
11. Maier D (1983) The theory of relational databases. Computer Science Press, Rockville
12. Mora A, Cordero P, Enciso M, Fortes I, Aguilera G (2012) Closure via functional dependence simplification. Int J Comput Math 89(4):510–526
13. Mora A, de Guzmán IP, Enciso M, Cordero P (2011) Ideal non-deterministic operators as a formal framework to reduce the key finding problem. Int J Comput Math 88(9):1860–1868
14. Pernelle N, Sas F, Symeonidou D (2013) An automatic key discovery approach for data linking. Web Semant Sci Serv Agents WWW 23:16–30
15. Saiedian H, Spencer T (1996) An efficient algorithm to compute the candidate keys of a relational database schema. Comput J 39(2):124–132
16. Sismanis Y, Brown P, Haas PJ, Reinwald B (2006) Gordian: efficient and scalable discovery of composite keys. In: Proceedings of the international conference on very large data bases, pp 691–702
17. Sridhar R, Iyengar SS (1990) Efficient parallel algorithms for functional dependency manipulations. In: Proceedings of the 2nd international symposium on databases in parallel and distributed systems, DPDS '90, pp 126–137

18. Valtchev P, Duquenne V (2003) Towards scalable divide-and-conquer methods for computing concepts and implications. In: 4th International conference Journèes de l'Informatique Messine (JIM03): knowledge discovery and discrete mathematics, Metz (FR), pp 3–14
19. Valtchev P, Duquenne V (2008) On the merge of factor canonical bases. In: International conference on formal concept analysis (ICFCA). Lecture notes in computer science, vol 4933. Springer, Berlin, pp 182–198
20. Wastl R (1998) On the number of keys of a relational database schema. J Univers Comput Sci 4:547–559
21. Worland PB (2004) An efficient algorithm for 3nf determination. Inf Sci Inf Comput Sci 167(1–4):177–192
22. Yu CT, Johnson DT (1976) On the complexity of finding the set of candidate keys for a given set of functional dependencies. Inf Process Lett 5(4):100–101
23. Zobel J (1998) How reliable are the results of large-scale information retrieval experiments? In: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '98, pp 307–314