Increasing the Efficiency of Minimal Key Enumeration Methods by Means of Parallelism

Fernando Benito[1], Pablo Cordero[2], Manuel Enciso[1] and Ángel Mora[2]
[1]*Department of Languages and Computer Science, University of Málaga, Málaga, Spain*
[2]*Department of Applied Mathematics, University of Málaga, Málaga, Spain*

Keywords: Functional Dependency, Minimal Key, Parallelism, Logic, Tableaux-method.

Abstract: Finding all minimal keys in a table is a hard problem but also provides a lot of benefits in database design and optimization. Some of the methods proposed in the literature are based on logic and, more specifically on tableaux paradigm. The size of the problems such methods deal with is strongly limited, which implies that they cannot be applied to big database schemas. We have carried out an experimental analysis to compare the results obtained by these methods in order to estimate their limits. Although tableaux paradigm may be viewed as a search space guiding the key finding task, none of the previous algorithms have incorporated parallelism. In this work, we have developed two different versions of the algorithms, a sequential and a parallel one, stating clearly how parallelism could naturally be integrated and the benefits we get over efficiency. This work has also guided future work guidelines to improve future designs of these methods.

1 INTRODUCTION

Identifying properly all the keys of a table in a relational schema is a crucial task for several areas in information management: data modeling (Simsion and Witt, 2005), query optimization (Kemper and Moerkotte, 1991), indexing (Manolopoulos et al., 1999), etc. Key constraints specify sets of attributes in a relation such that their projection univocally identifies each tuple of the relation. Therefore, each key is composed by a subset of attributes playing the role of a *domain* in a given function whose *image* is the whole set of attributes. This way, the table is viewed as its extensional definition. These functions are described by means of a *Functional Dependency (FD)* which specifies a constraint between two subset of attributes, denoted $A \to B$, ensuring us that for any two tuples in a table, if they agree on A, they also agree on B.

All functional dependencies satisfied in a given table may be deduced from its dataset using data mining techniques (Appice et al., 2011; Huhtala et al., 1999), or may be provided by database designers. It is out the scope of this work to extract FDs from relational tables, since it becomes a data mining problem (Fayyad et al., 1996). Minimal key problem consists in finding all the attribute subsets which make up a minimal key given a set of FDs occurring within a schema of a relational table.

Nowadays, several algorithms are capable to solve this problem using different classical techniques (Lucchesi and Osborn, 1978; Yu and Johnson, 1976; Saiedian and Spencer, 1996; Zhang, 2009; Armstrong, 1974) (see Section 2 for further details). Recently, some alternative methods have been introduced using logic. In this work we will concentrate on algorithms guided by logic, and most specifically, those using tableaux paradigm (Morgan, 1992) for deriving keys of a relation schema using inference systems. As we shall see later, tableaux might be considered a flexible and powerful framework to design automated deduction methods to solve complex problems in a flexible and effective way.

In previous works, several tableaux-like methods have been introduced (Wastl, 1998a; Cordero et al., 2013). Efficient versions of both of them have been implemented. Nevertheless, tableaux-like methods generate wide search spaces and, in many cases, the same solution (same minimal key) appears at the end of several branches of the tree representing the search space. These intrinsic characteristics of tableaux-like methods establish a strong limitation in the size of the problems to be treated by them and, usually, discourage their use.

Indeed, sequential implementation of these methods produces such an explosion of search space that we go beyond machine capabilities, even with small

Benito Picazo F., Cordero P., Enciso M. and Mora Á..
Increasing the Efficiency of Minimal Key Enumeration Methods by Means of Parallelism.
DOI: 10.5220/0005108205120517
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 512-517
ISBN: 978-989-758-036-9

problems (20+ FDs for $\mathbb{K}$'s method (Wastl, 1998a) as it has been demonstrated in previous studies of this work (Cordero et al., 2013)). However, a very interesting property within search spaces induced by tableaux methods, is the fully independence of their branches, so we can directly consider them as separated sub-problems leading to several solutions of the original problem. It is in this spirit that tableaux paradigm supplies us an optimal guide to build parallel algorithms finding all minimal keys in a table, since the building of the tableaux tree directs the parallel and independent processing.

In this work, we have developed parallel implementations of tableaux-like methods to solve minimal keys finding problems. As shown in the following, they have significantly increased the size of the problem these methods are able to handle. Intentionally, we have executed them under different hardware configurations trying to discover tendencies in which the efficiency within method could be influenced. As already implied above, a processing cluster will be available for us to engage problems with a substantial size at the input so we can deal with them in terms of storage capacity and execution time.

The paper is organized as follows: In Section 2 we introduce several background. A brief study of the state-of-the-art is exposed in Section 3. Section 4 introduces us into sequential tableaux-like methods showing some experimental results. Parallelism implementations enter the scene in Section 5 presenting their way to proceed, and showing the high benefits obtained. Several conclusions are then given in Section 6.

2 BACKGROUND

We begin this section with three brief definitions of basic concepts.

Definition 1 (Functional dependency)**.** *Let U be a set of attributes. A functional dependency (FD) over U is an expression of the form $X \rightarrow Y$, where X, Y are attribute sets. 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 a subset of attributes that allows us to uniquely characterize each row. It may be defined by means of functional dependencies as follows:

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

Definition 3 (Minimal Key)**.** *Given the table R, the attribute set $K \subset U$ is said to be a minimal key if it is a key of R and for all attribute $k \in K$ the subset $K - \{k\}$ is not a key of R.*

Due to space limitation, we refer those readers non familiar with the formal notions of FDs, keys and relational tables to (Elmasri and Navathe, 2010). In Table 1, we just illustrate its semantics by a basic example.

From the information in Table 1, we may ensure that the following FDs are satisfied: $Title, Year \rightarrow Country$, $Title, Year \rightarrow Director$ and $Director \rightarrow Nationality$. Moreover, the table has only one minimal key: $\{Title, Year, Star\}$

Inferring minimal keys from a set of FDs has been well studied. The algorithm of Lucchesi and Osborn (Lucchesi and Osborn, 1978) to find all the keys in a relational schema is considered the first deep study around this problem. Yu and Johnson (Yu and Johnson, 1976) established that the number of keys is limited by the factorial of the number of dependencies, so, there does not exist a polynomial algorithm for this problem.

3 TABLEAUX-LIKE METHODS

Some authors have used several techniques to solve this problem. Saiedian and Spencer (Saiedian and Spencer, 1996) propose an algorithm for computing the candidate keys using attribute graphs. Zhang in (Zhang, 2009) uses Karnaugh maps to calculate all the keys. Nevertheless, we are interested in the use of artificial intelligence techniques and, more specifically, in the use of logic. Armstrong's axiomatic system (Armstrong, 1974) is the former system introduced to manage FDs in a logic style. In (Wastl, 1998b), for the first time a Hilbert-styled inference system for deriving all keys of a relation schema was introduced. Alternatively, in (Cordero et al., 2013) the authors tackle the key finding problem with another inference rule inspired by the Simplification Logic for Functional Dependencies. These two papers constitute the target algorithms to be compared in this work. We refer the reader to the original papers for further details.

Both works are strongly based on tableaux paradigm. Tableaux-like methods represent the search space as a tree, where its leaves contain the solutions (keys). Tree building process begins with an initial root and from there on, inference rules generate new branches labeled with nodes representing simpler instances of the parent node. The very best advantage of this process goes to its versatility, since developing new inference systems –which is not a trivial task indeed– allows us to design a new method. Com-

Table 1: Movie table.

Title	Year	Country	Director	Nationality	Star
Pulp Fiction	1994	USA	Quentin Tarantino	USA	John Travolta
Pulp Fiction	1994	USA	Quentin Tarantino	USA	Uma Thurman
Pulp Fiction	1994	USA	Quentin Tarantino	USA	Samuel L. Jackson
King Kong	2005	New Zealand	Peter Jackson	New Zealand	Naomi Watts
King Kong	2005	New Zealand	Peter Jackson	New Zealand	Jack Black
King Kong	1976	USA	De Laurentiis	IT	Jessica Lange
King Kong	1976	USA	De Laurentiis	IT	Jeff Bridges
Django Unchained	2012	USA	Quentin Tarantino	USA	Jamie Foxx
Django Unchained	2012	USA	Quentin Tarantino	USA	Samuel L. Jackson

parisons between tableaux-like methods can be easily drawn as its efficiency goes hand-in-hand with the size of the generated tree.

Although $\mathbb{K}$ and SL_{FD} are the two inference systems which are the basis of two tableaux-like methods, they are very different. $\mathbb{K}$ is the former basis for a Hilbert-styled method and it deals with unitary functional dependencies, which produce a significant growth of the input set. SL_{FD} avoids the use of fragmentation by using generalized formulas. It is guided by the idea of simplifying the set of FDs by removing redundant attributes efficiently. Moreover, SL_{FD} incorporates a pre-processing task at a first step which prunes the input up to an algebraic characterization of the problem by providing significant reduction of the problem's size to be treated by the tableaux-like method, which is the hardest task.

4 LIMITS OF SEQUENTIAL TABLEAUX-LIKE METHODS

As we mentioned in the introduction, in this section we show the strong limitation that sequential implementation of tableaux-like methods will face to solve minimal keys.

We have developed two efficient implementations of both methods and they have been executed over a high performance architecture sited in the Supercomputing and Bioinnovation Center [1]. In these experiments we take into account the following parameters: execution time in seconds (named [Ti]), number of nodes in the tableaux search space (named [No]) and number of redundant keys (named [RK]). This last parameter shows the impact of extra branches, i.e., the number of duplicated keys computed by the algorithm.

Execution times may be considered a parameter linked to the architecture we are using for running the experiments. Number of nodes and redundant keys represent the size of the search space and the repeated operations respectively. They are independent of the implementation, providing a fair comparison between methods in the future.

4.1 First Experiment: Benchmarking Problems

Although there not exists a benchmarking for functional dependency problems, our first intention was to execute the methods over a set of different and characteristic problems. Thus, we began building a battery of problems gathered from several related papers around (Saiedian and Spencer, 1996; Wastl, 1998a) conforming an initial suitable set of problems in a benchmarking style. Results obtained for this battery of problems are shown in Table 2.

As shown in Table 2 only one of the entry problems needs more than one second to finalize and it is in the case of $\mathbb{K}$ method. This is due to the small sets handled by these academic problems. Results for SL_{FD} are better than $\mathbb{K}$ except for saiedian3 problem. Indeed, there are cases where we need just one node to finish the algorithm, corresponding to those problems where the algebraic characterization used by SL_{FD} reduces the problem to its canonical version.

4.2 Second Experiment: Random Problems

In this subsection we deal with larger randomly generated problems. We have built several examples varying two parameters: number of attributes and number of FDs. We have not established a correlation between both parameters in the generation because different ratios between them produce problems with significant differences. Moreover, observe that number of minimal keys is not directly influenced by these two parameters.

The size of the problems in Table 3 is greater than those presented in previous section. They may be considered *medium*-size problems, having param-

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

Table 2: Sequential executions over benchmarking problems.

Problem	Attrib	FDs	Keys	$\mathbb{K}$			SL_{FD}		
				[Ti]	[No]	[RK]	[Ti]	[No]	[RK]
saiedian1	6	5	1	0	12	5	0	1	0
saiedian2	6	5	3	0	7	0	0	10	3
saiedian3	7	7	4	0	139	64	0	674	284
derivation5	9	4	5	0	17	4	0	41	20
a3rojo	7	5	2	0	81	29	0	4	0
elmasri1	6	3	1	0	1	0	0	1	0
wastl2	7	3	1	0	2	0	0	1	0
wastl10	3	3	1	0	5	2	0	1	0
wastl13	4	4	3	0	10	2	0	13	5
example001	10	7	8	20	55.768	24.174	0	1.090	448

Table 3: Sequential executions over random problems.

Problem	Attrib	FDs	Keys	$\mathbb{K}$			SL_{FD}		
				[Ti]	[No]	[RK]	[Ti]	[No]	[RK]
med1	10	10	3	155	1.463.228	723.372	0	902	404
med2	5	17	4	0	1.906	1.029	0	1.149	772
med3	15	7	2	40	432.104	220.230	0	143	71
med4	30	5	5	12	802.770	300.485	12	23	7
med5	20	10	3	180	2.038.746	1.012.651	3	1.102	666
med6	5	50	5	20	218.892	179.444	1	129.508	117.461
med7	40	10	2	204	1.130.539	467.512	0	122	53
med8	15	15	7	38	6.715.949	3.023.693	5	77.922	41.070
med9	7	50	5	325	32.219.336	21.357.930	18	2.760.961	2.227.596
med10	10	20	4	835	496.380.119	218.275.528	8	20.442	9.966

eters which properly match with real tables in software engineering and execution times are quite reasonable, particularly for SL_{FD} method (less than one minute). Nevertheless, we notice that as far as problem's size grows (even just a little), results go considerably beyond. Therefore, we are absolutely limited when dealing with real problems, where the input size would be worthy of consideration. So the challenge is to figure out whether parallelism will help us solving these kind of problems, and even greater ones.

5 PARALLELIZATION OF TABLEAUX-LIKE METHODS

As a conclusion of the experiments presented in previous section, parallel strategy and big hardware resources will be totally indispensable if we want to compare tableaux-like methods from one to another. Our intention is to take advantage of tableaux design to split the original problem into *atomic* instances that may be solved within a reasonable time and resources. Then, we combine the solutions of all these sub-problems to enumerate all the minimal keys.

Thus, our parallel implementations of the algorithms will work in two steps:

1. Splitting step: It executes the tableaux-like methods but it will stop in a determinate level that we will introduce at the process call, generating a set of sub-problems. The level is induced by the size of the root (the number of attributes in this level).
2. Parallel step: In this second stage we execute parallel task solving those sub-problems and, at the end, we combine all the solutions to get all minimal keys.

In order to test parallel versions we run another battery of problems whose results are retrieved in Table 4. This time we include several new columns gathering parallel implementation's parameters: Break-off value [L] and number of generated sub-problems [Sp].

It is imperative to state some critical considerations concerning the limit size of the atomic problems. In one hand, we have observed that the greater this limit is, the better will be the improvement by parallelism, since it *would* generate a higher number of sub-problems. However, as far as we try to make this improvement to be better by a wide limit value, the longer the partial version will take to split the entry problem.

In addition, this is not an independent parameter among the algorithms, we need to choose a different break-off value depending on the method we are using as each one of them will need a particular amount of resources.

Table 4: Parallel executions over random problems.

Prob	Attr	FDs	Keys	$\mathbb{K}$					SL_{FD}				
				[L]	[Sp]	[Ti]	[No]	[RK]	[L]	[Sp]	[Ti]	[No]	[RK]
cp01	7	50	5	6	87	0	32.219.336	21.357.930	45	50	3	2.760.961	2.227.596
cp02	10	20	4	9	44	0	496.380.119	218.275.528	10	268	11	20.442	9.966
cp03	15	20	3	11	32	1	17.917.662	9.340.225	15	78	4	1.405.153	814.026
cp04	20	20	4	10	27.284	31	3.145.751.761	1.424.991.475	15	47	3	587.765	313.513
cp05	20	30	45	12	1.696	3	1.563.813.853	677.457.455	24	98	136	271.402.277	162.828.760
cp06	10	35	5	8	2.358	3	121.396.806	65.571.971	30	158	8	3.215.995	1.686.149
cp07	25	15	15	14	25.836	28	3.975.400.144	1.980.982.101	8	802	39	220.047	914.80
cp08	15	40	1	9	39.708	46	837.341.359	433.068.418	0	0	1	1	0
cp09	25	20	25	14	33.146	38	123.283.772.804	59.975.556.886	12	18.999	1.077	47.014.652	23.418.562
cp10	35	10	37	22	1.370	5	101.429.265.443	68.138.197.993	7	219	10	27.985	13.436

As an example of this last point, we would like to check execution times for cp09 and cp10 in the case of $\mathbb{K}$, where the huge difference is due to a wiser splitting process.

As a general conclusion, execution times are pretty reasonable considering the dimension of these entry problems (several minutes were enough to resolve most of cases).

On a separate issue, we can notice that results are pretty huge for this kind of problems using $\mathbb{K}$'s method. The hugest number of nodes of the tableaux overtakes up to 123 billions of nodes. So, efficiency of $\mathbb{K}$'s method is so far to be accepted. SL_{FD} reaches better times and dimensions tableaux in the very most of cases. Thus, several noteworthy outcomes come up.

First, problem cp04 needs a tableaux of over half-million nodes with SL_{FD}, while $\mathbb{K}$ goes up to 3 billions! This is due to the high number of FDs after the hard fragmentation rule inherent to $\mathbb{K}$. A similar situation involves cp09 and cp10 problems.

Finally, if we care about useless computing time, we notice the number of redundant keys is terrible; the difference here between both methods is as impressive as in the rest of parameters. For instance, the resulting set of minimal keys for cpx04 problem contains just 4 minimal keys. However, $\mathbb{K}$ generates 1,424,991,475 redundant keys and SL_{FD} 313,513. This is indeed, a huge waste of space and time.

6 CONCLUSIONS

The first point we want to state clear is that the concept of parallelism we are dealing with refers to a *hardware parallelism*. We mean that the benefits we are obtained from parallelism are due to a cluster of cores where we can deliver each of our jobs continuously.

In order to resolve real problems where the size of the input is substantial, it is imperative to count on the participation of a great amount of resources. Moreover, it is difficult, at a first look, to estimate whether a given problem will result in a difficult or an easy one. In some sense we may say that it is a chaotic and unpredictable problem.

A glance is enough to easily realize that $\mathbb{K}$ algorithm needs more time, more nodes and more redundant keys to reach the solution than SL_{FD} in the very most of cases. Indeed, the differences are not trivial so far. Concerning the size of the tableaux, $\mathbb{K}$ builds up billion of nodes whereas SL_{FD} generates a reasonable amount of nodes. A similar conclusion may be established for the number of redundant keys.

Finally, establishing an appropriate limit to split up the entry problem in the parallel versions of the algorithms is not an easy issue so far. We have to run several experiments to reach a good value which will not spend so much time splitting the entry but it should spend time enough to take advantage of the parallelism.

ACKNOWLEDGEMENTS

Supported by Grant TIN2011-28084 of the Science and Innovation Ministry of Spain.

REFERENCES

Appice, A., Ceci, M., Turi, A., and Malerba, D. (2011). A parallel, distributed algorithm for relational frequent pattern discovery from very large data sets. *Intell. Data Anal.*, 15(1):69–88.

Armstrong, W. W. (1974). Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583.

Cordero, P., Enciso, M., and Mora, A. (2013). Automated reasoning to infer all minimal keys. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 817–823. AAAI Press.

Elmasri, R. and Navathe, S. (2010). *Fundamentals of Database Systems*. Prentice Hall International, 6 edition.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, pages 37–54.

Huhtala, Y., Krkkinen, J., Porkka, P., and Toivonen, H. (1999). Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111.

Kemper, A. and Moerkotte, G. (1991). Query optimization in object bases: Exploiting relational techniques. In *Query Processing for Advanced Database Systems, Dagstuhl*, pages 63–98. Morgan Kaufmann.

Lucchesi, C. L. and Osborn, S. L. (1978). Candidate keys for relations. *J. Comput. Syst. Sci.*, 17(2):270–279.

Manolopoulos, Y., Theodoridis, Y., and Tsotras, V. J. (1999). *Advanced Database Indexing*, volume 17 of *Advances in Database Systems*. Kluwer.

Morgan, C. G. (1992). An automated theorem prover for relational logic (abstract). In Fronhfer, B., Hhnle, R., and Kufl, T., editors, *TABLEAUX*, pages 56–58.

Saiedian, H. and Spencer, T. (1996). An efficient algorithm to compute the candidate keys of a relational database schema. *Comput. J.*, 39(2):124–132.

Simsion, G. C. and Witt, G. C. (2005). *Data modeling essentials*. Amsterdam; Boston, 3rd edition.

Wastl, R. (1998a). Linear derivations for keys of a database relation schema. *J. UCS*, 4(11):883–897.

Wastl, R. (1998b). On the number of keys of a relational database schema. *Journal of Universal Computer Science*, 4.

Yu, C. T. and Johnson, D. T. (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.

Zhang, Y. (2009). Determining all candidate keys based on karnaugh map. *IEEE International Conference on Information Management, Innovation Management and Industrial Engineering*, 04:226–229.