

Fast Mining of Association Rules in Large-Scale Problems

Hussien H. Aly
Computers and Automatic
Control Dept. Alexandria Univ.,
Alexandria, Egypt
alyh@computer.org

Ashraf A. Amr
IRI, Mubarak city for SRTA ,
Alexandria, Egypt
ashraf.amr@usa.net

Yousry Taha
Computers and Automatic Control
Dept. Alexandria Univ.,
Alexandria, Egypt
yousry_taha@hotmail.com

Abstract

In this paper we study scalability problem of apriori – like algorithms that are used in mining association rules. We show how apriori suffers form performance deterioration for large-scale problems and propose an alternative data structures and operations that can be used to apply the apriori-trick optimization method in large-scale problems.

In the proposed method, the database is transformed into a more efficient structure that is used along with the intersection operation, to find the frequent itemsets in the database. The performance evaluation shows that, with a minor increase in the storage requirement, the proposed technique outperforms significantly the existing algorithms especially in large-scale problems.

1. Introduction

Since the introduction of association rule mining and the a-priori algorithm [1,2], several optimizations and extensions to the basic algorithm have been proposed in the literature such as [3-10]. A recent survey of the mining problem can be found in [11]. Integration of the association rule mining with the relational database technology and the SQL is studied in [13,14,15]. This technology is now used to discover and analyze the relationship between objects in new areas of applications for example, between Web pages that is implied by the users' accessing behavior. The ever-increasing size of log files in these applications makes it very difficult for conventional methods of mining to be applied.

The main success of these conventional methods in small and medium-sized databases is usually attributed to the so-called "a-priori trick" [12] optimization which reduces the search space for frequent sets of items (itemsets). Using the terminology of the well-known market-basket analysis problem [11], we can state this optimization method as follows: "If an itemset is not frequent (its count is less than a minimum support value) in a set of transactions, then any superset of this itemset is

not frequent in the same set of transactions".

The rest of this paper is organized as follows. We discuss the scalability and performance problems of apriori in Section 2. In Section 3, the inverted algorithm, the data structure and the intersection operation are presented. In Section 4, the performance evaluation, the scalability and comparison with other related existing algorithm are given. Finally Section 5 gives our conclusion.

2. Apriori scalability problem

The mining problem size is usually determined by four parameters: number of transactions, number of items, the transaction size and the threshold. The experiments performed in [2] studied each parameter of the problem individually. For example, experiments are performed with different number of items keeping the number of transactions fixed (at 100,000 transactions). And so on for the other parameters.

Unfortunately, these parameters which define the problem size are interacting in a non-uniform way. Increasing one parameter while keeping some other parameters fixed not necessarily increase the problem size. For example, increasing the number of items and fixing both the threshold and the number of transactions will lead to a smaller number of candidate and frequent itemsets and even makes the algorithms to finish in fewer iterations. On the other hand, increasing both the number of items and the number of transactions or increasing the number of items and lowering the threshold level may lead to the crash of the algorithms due to exponential explosion.

In this paper we studied this scalability problem and show how apriori suffers form performance deterioration for large-scale problems. We also propose an alternative data structures and operations that can be used to apply the apriori-trick optimization method in large-scale problems. We called this method the *inverted method* since it depends on inverting the transactional database structure. The inverted method is an apriori-like level-wise technique and it is important to notice that it is orthogonal to other optimization techniques that are applied to apriori. In other

words, we can apply these techniques also to the inverted method.

3. Transactional database inversion

Transactional databases contain transactions such as those found in the retail environments where sales transactions are stored in the database or those found in Web environments where the log files record information about pages that are requested. The information stored in such databases includes the transaction ID and the set of items bought in this transaction (or the set of pages visited in the web site). In addition, it may contain a time stamp (time and date of the transaction), customer ID and other information about the transactions. The usual representation of the transactional data is written as $\{\text{TransID}; \{\text{ItemID}_1, \text{ItemID}_2, \dots, \text{ItemID}_n\}\}$

Note that the above is the minimum information that can be stored for the transaction. The actual data format can be any arbitrary format, for example, relational data or just sequential file. All apriori-like algorithms that are currently used or proposed for data mining transactional databases directly use this format or a very close one. Scanning the whole database at each iteration of apriori is one of the main contributors to the time required for generating the frequent itemsets.

In this paper, we propose a transformation that inverts the above representation by storing the data item-wise not transaction-wise. A record of the transformed database can be written as $\text{ItemID}; \{\text{TransID}_1, \text{TransID}_2, \dots, \text{TransID}_m\}$

In the first iteration, we use the original database to construct the inverted one while searching for the candidate frequent items. In higher iterations (starting from iteration 2) of the association rule mining, we will use this inverted representation. That is a database record in the k^{th} iteration will have the form $\{\text{Itemset}; \text{Transactionset}\}$. Where *Itemset* is the candidate set of items and *Transactionset* is the set of transactions in which all the members of the itemset appear. The size of the transformed database is roughly equal to the size of the original database.

3.1. Mining using the transformed database

To perform association rule mining using the transformed database, we use the intersection operation. The idea is explained in the following theorem. The complete proof of the theorem can be found in [16].

Theorem: Consider two records of the form (1) above; then the transactionset for the union of the two itemsets is the intersection of their transactionsets.

Using the same level-wise technique as the apriori algorithm and utilizing the above theorem to implement

the apriori trick we can perform the association rule mining as follows:

Start with the original transactional database, transform it into the inverted structure. This can be done in a single scan of the database. An item is frequent if the cardinality of its transactionset is higher than the required minimum frequency. So, we can get the frequent 1-itemsets. All the following iterations that generate higher candidate itemsets will use this inverted structure. To verify if a candidate k -itemset is frequent or not, perform the intersection operation for any two of its $(k-1)$ -subsets. If the cardinality of the resulting transactionset is higher than the required minimum frequency, the itemset is frequent. Otherwise, it is not.

Note that there are many optimization issues involving the intersection operations. For example, we can choose from $(3^k - 2^{k+1} + 1)/2$ different alternative operands for the intersection operation (dividing the k -itemset into two not necessarily disjoint subsets). Which alternative to choose is an optimization decision to minimize the cost of the intersection operation. Since the transactionset size decreases as the itemset cardinality increases, we may choose a division of the k -itemset of interest into two $k-1$ -subsets to reduce the cost of the intersection operation. This is due to the fact that the transactionset for a k -itemset is a subset of that of any of its subsets. We may also choose the two with the minimum transactionset size. This depends on the materialization strategies we use. Due to space limitation, we refer the readers to [16] for the complete and formal description of the algorithm.

4. Performance evaluation and comparison

The performance experiments are performed on an Intel PIII processor with 450MHz CPU clock and 64MB memory running windows 98. The machine uses the FAT file system with measured throughput of 2.7 MB/sec.

The synthetic data generation program developed in [2] is used to generate different datasets of various characteristics. The parameter definitions used by the synthetic data generator are as follows: N is the number of items, $|D|$ is the number of transactions, $|T|$ is the average transaction length, $|I|$ is the average length of the potentially large itemsets used to generate the datasets.

We first generated several datasets with the same parameters as those in [2] in order to verify our simulator and compare the result with the known published results. These datasets represent relatively small-to-medium problem sizes. Then, we increase the problem size and measure the performance of different methods to see the scalability of these methods.

Table 1 gives the characteristics of the first generated datasets. Note that all datasets have the same number of transactions ($|D|=100,000$) and number of items ($N=1000$) as specified in [2]. The dataset T5.I2 means that the

average transaction length is 5 items and the average length of the potentially frequent itemset used in the generating procedure is 2. The other columns give the total size, the size of the inverted structure and the time required for inversion. In this paper, due to space limitation, we shall present the results of some of these datasets. Please refer to [16] for a complete presentation.

Table 1. The generated datasets-1

Dataset	T	I	Size (MB)	Inverted size (MB)	Inversion time (sec)
T5.I2	5	2	2.6	1.9	2.03
T10.I2	10	2	4.5	3.8	3.68
T10.I4	10	4	4.5	3.8	3.68
T20.I2	20	2	8.4	7.6	9.83
T20.I4	20	4	8.4	7.6	9.94
T20.I6	20	6	8.4	7.6	9.98

4.1. Execution time

Figure 1 gives the average total execution time comparison between the two techniques for T5.I2, T10.I4, and T20.I6 with different thresholds. The database inversion time of the datasets is not included in the mining time of the inverted method since it is done only once for each of the generated datasets, independent of the threshold values.

The curves of Figure 1 show that, in general, the inverted method slightly outperforms the apriori algorithm and both have the same behavior when using smaller threshold values. The difference increases when the problems size increase.

An important variant of apriori called “aprioriTid” is also presented and studied in [2] and is shown there to have better performance than apriori in higher iterations only. However, in early iterations aprioriTid performs very slowly than apriori. We studied the iteration time for the apriori, aprioriTid and the inverted method. The results are shown in Figure 2, where it shows that the iteration times for the inverted method is less than the corresponding ones of apriori and aprioriTid. For higher iterations, the inverted method is not worse than the aprioriTid.

4.2. Number of transactions scale-up

In this experiment we test the effect of changing the number of transactions on the performance of the algorithms. The number of transactions is varied in the range from 100,000 to 1 million. The results are given in Figure 3 and show that the inverted method beats the apriori algorithm in all cases. The difference increases as the database size increases.

4.3. Number of items scale-up

In this experiment we test the effect of changing the number of items on the performance of the algorithms. The number of items is tested in the

range from 1000 to 10000. The results are given in Figure 4. It is important to note that increasing the number of items with the number of transactions fixed at 100,000 reduces the number of candidates and frequent itemsets and even makes the algorithms finishes in few iterations (about 2 iterations in almost all cases).

Again, the results indicate that the two algorithms have the same trend behavior. However, the inverted method takes about 50% of the time taken by apriori.

4.4 Transaction size scale-up

In this experiment we test the effect of changing the transaction size (number of items per transactions) on the performance of the algorithms. The transaction size ranges from 5 to 50. To avoid the effect of other parameters, we vary the number of transactions as well such that the total database size is the same for all cases. The number of transactions ranges from 200,000 for transaction size of 5 to 20,000 for transaction size 50. The results are given in Figure 5. Note that the thresholds are not set to percentage of the number of transaction as the other experiments, rather they set to absolute values. This is because the number of transactions varies with the transaction size as stated above.

As shown in Figure 5, the inverted method scales very well in this case. The apriori algorithm scales linearly.

4.5 Problem size scale-up

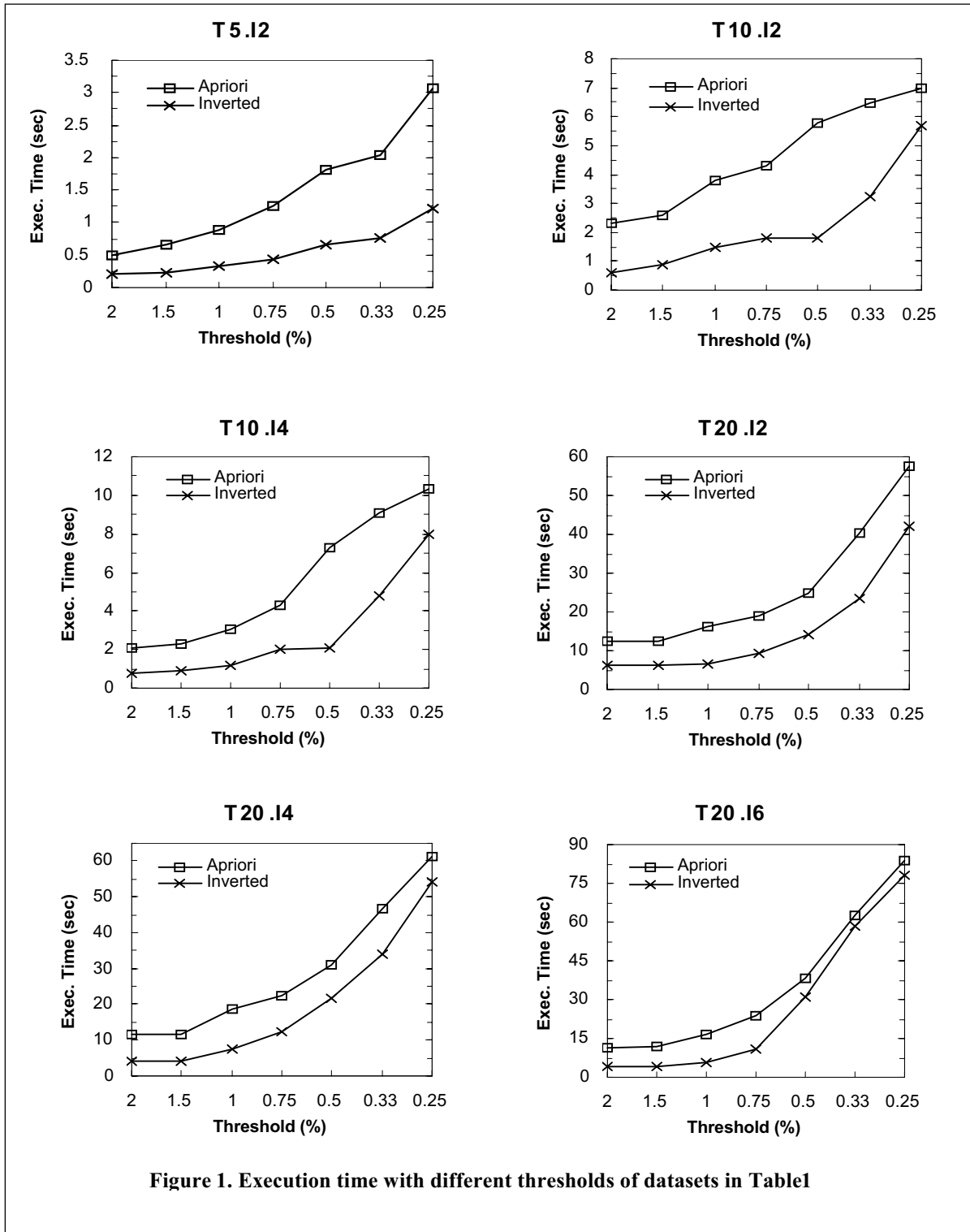
In this experiment, we study the effect of increasing the problem size on the execution time. As discussed before, the problem size increases by varying a combination of the problem parameters. We generated larger datasets with number of transactions of 1 million, rather than 100,000 as in the previous experiments. Also the number of items ranges from 1000 to 10000. The algorithms are tested with different threshold values. The database sizes in this experiment are nearly equal. They are about 26 MB. The inverted database size is about 19 MB. The inversion time is about 33 seconds.

The results are given in Figure 6. It shows that the inverted method beats apriori by order of magnitude for larger problems, especially for low thresholds. This behavior has many reasons. For large-scale datasets with large number of items, the number of candidate and frequent itemsets becomes very large. For example, in some cases, the number of candidate 2-itemsets was about 10 millions. Also for low thresholds, the number of iterations or passes over the database is increased as well.

As shown in Figure 6, apriori could not handle the case for low threshold values and large number of items (>7500). Apriori has to generate all candidate itemsets and then scanning the database to verify for each candidate itemset that all members are indeed in the transaction and count its frequency. The inverted method

does not have to generate all candidates at once or scan the database. It tests every candidate itemset directly since all the information needed in one single record.

We can see from Figure 6 that the inverted method scales very well with problem sizes. Note that some apriori curves of Figure 6 are not completed. These are usually



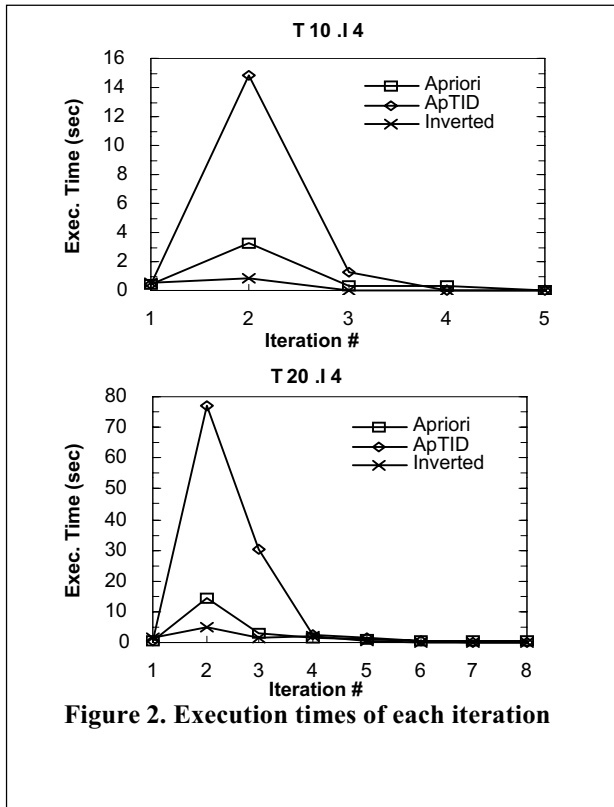


Figure 2. Execution times of each iteration

some runs with low threshold. These runs were aborted because they tended to take too long time. The runs with higher thresholds already show the trends of these curves.

5 conclusion and future work

In this paper, we introduce the inverted method for the discovery of frequent itemsets of a transactional database. The proposed method is level wise like apriori and it uses the apriori property for candidate generation. However, the counting operation of itemsets is based on a database transformation and the intersection operation. The performance evaluation of the proposed method shows that it outperforms the existing algorithms. The difference is by order of magnitude for large-scale datasets and for low support thresholds. The experiments also show high scalability of the inverted method with number of items, number of transactions, transaction size.

The study of the interactive techniques, caching of previous results and incremental mining techniques in the context of the inverted method can be a good point for further research and study.

References

- [1] R. Agrawal, T. Imielinski, A. Swami, "Mining Association Rules between Sets of Items in Large Databases", Proc. ACM SIGMOD Int'l Conf. on Management of Data, SIGMOD'93 Washington D.C., May 1993, pp. 207-216.
- [2] R. Agrawal, and R. Srikant, "Fast algorithms for mining Association Rules", Proc. 1994 Int'l Conf. Very Large Data Bases, Santiago, Chile, Sept. 1994, pp. 487-499.
- [3] J. Park, M. Chen, and P. Yu, "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules", IEEE Trans. on Knowledge and Data Engineering, Vol. 9, No. 5, Sept./Oct. 1997, pp. 813-825.
- [4] D. Cheung, J. Han, V. Ng, and C. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique", Proc. 1996 Int'l Conf. Data Engineering, New Orleans, Louisiana, USA, Feb. 1996.
- [5] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases", Proc. 1995 Int'l Conf. VLDB, Zurich, Switzerland, Sept. 1995, pp. 432-443.
- [6] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases", Proc. 1995 Int'l Conf. VLDB, Zurich, Switzerland, Sept. 1995, pp. 420-431.
- [7] R. Srikant and R. Agrawal, "Mining generalized Association Rules", Proc. 1995 Int'l Conf. VLDB, Zurich, Switzerland, Sept. 1995, pp. 407-419.
- [8] V. Ganti, J. Gehrke, R. Ramakrishnan, "Mining Very Large Databases", IEEE Computer 1999 special edition on Data Mining, pages 38-45.
- [9] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, A. Rosenthal, "Query Flocks: A Generalization of Association-Rule Mining", Proc. ACM SIGMOD Int'l Conf. on Management of Data, Seattle, WA, USA, June 1998.
- [10] R. Agrawal and J. Shafer, "Parallel Mining of Association Rules", IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 6, Dec. 1996, pp. 962-969.
- [11] T. Shintani and M. Kitsuregawa, "Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy", Proc. ACM SIGMOD Int'l Conf. on Management of Data, SIGMOD'98 Seattle, WA, USA, June 1998.
- [12] E. Han, G. Karypis, V. Kumar, "Scalable Parallel Data Mining for Association Rules", Proc. ACM SIGMOD Int'l Conf. on Management of Data, SIGMOD'97 Tucson, Arizona, USA, May 1997, pp. 277-288.
- [13] S. Sarawagi, S. Thomas, R. Agrawal, "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications", Proc. ACM SIGMOD Int'l Conf. on Management of Data, Seattle, WA, USA, June 1998.
- [14] S. Thomas, S. Sarawagi, "Mining Generalized Association Rules and Sequential Patterns Using SQL Queries", Proc. 4th Int'l Conf. on Knowledge Discovery in Databases and Data Mining, New York, August 1998.
- [15] R. Agrawal, K. Shim, "Developing Tightly Coupled Data Mining Applications on a Relational Database System", Proc. 2nd Int'l Conf. on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August 1996.
- [16] Ashraf A. Amr, "Fast Mining of Interesting Association Rules for Large-Scale Problems" M.Sc. thesis, Alexandria University, Computers & Automatic Control Dept., Jan. 2001.

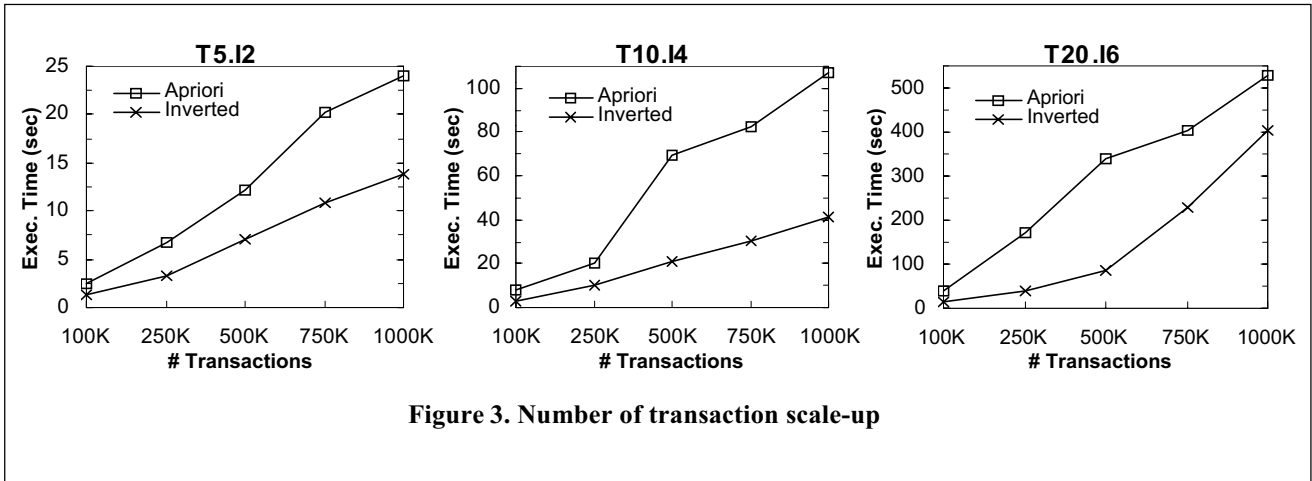


Figure 3. Number of transaction scale-up

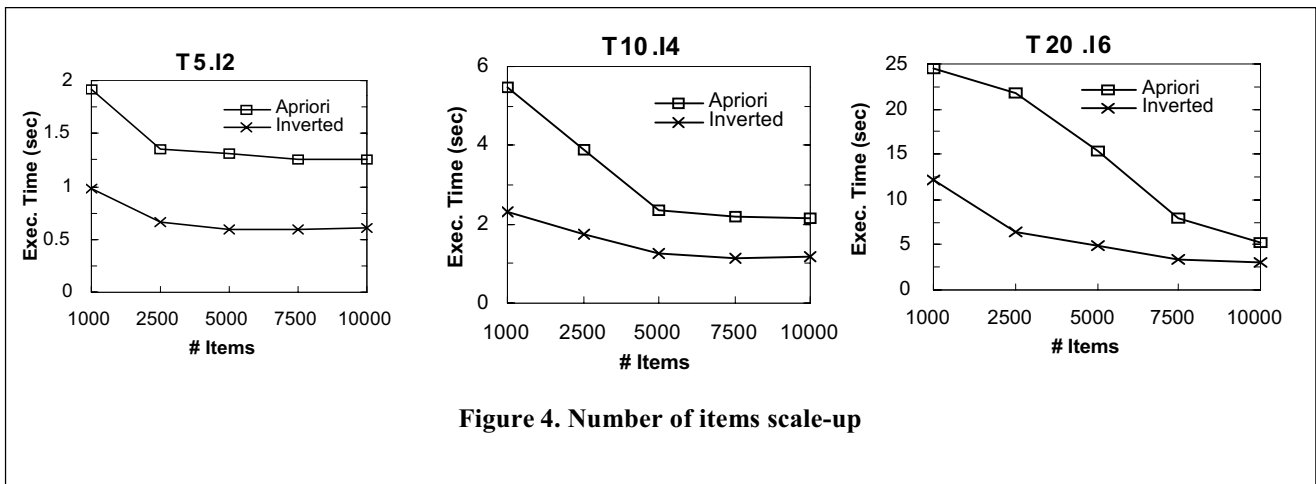


Figure 4. Number of items scale-up

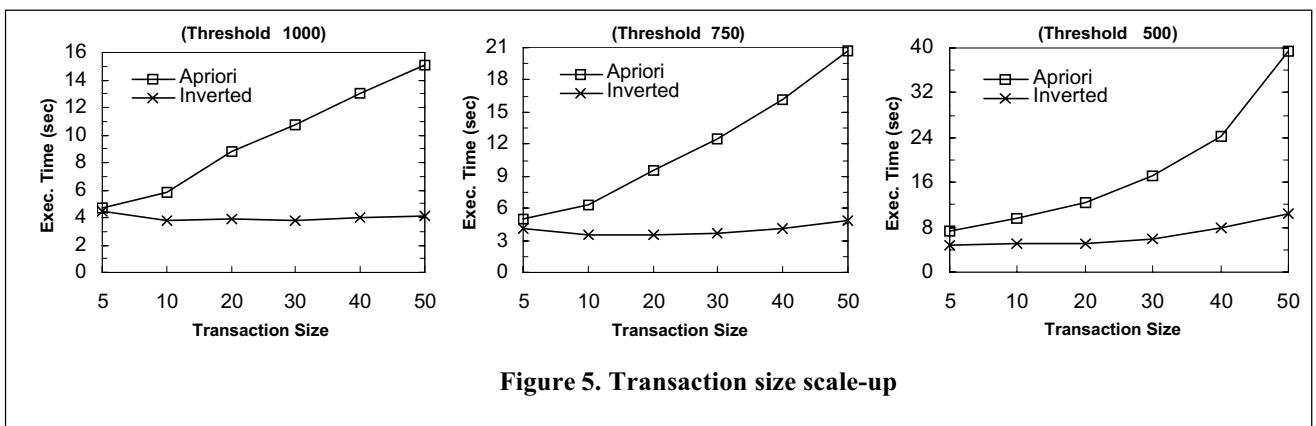


Figure 5. Transaction size scale-up

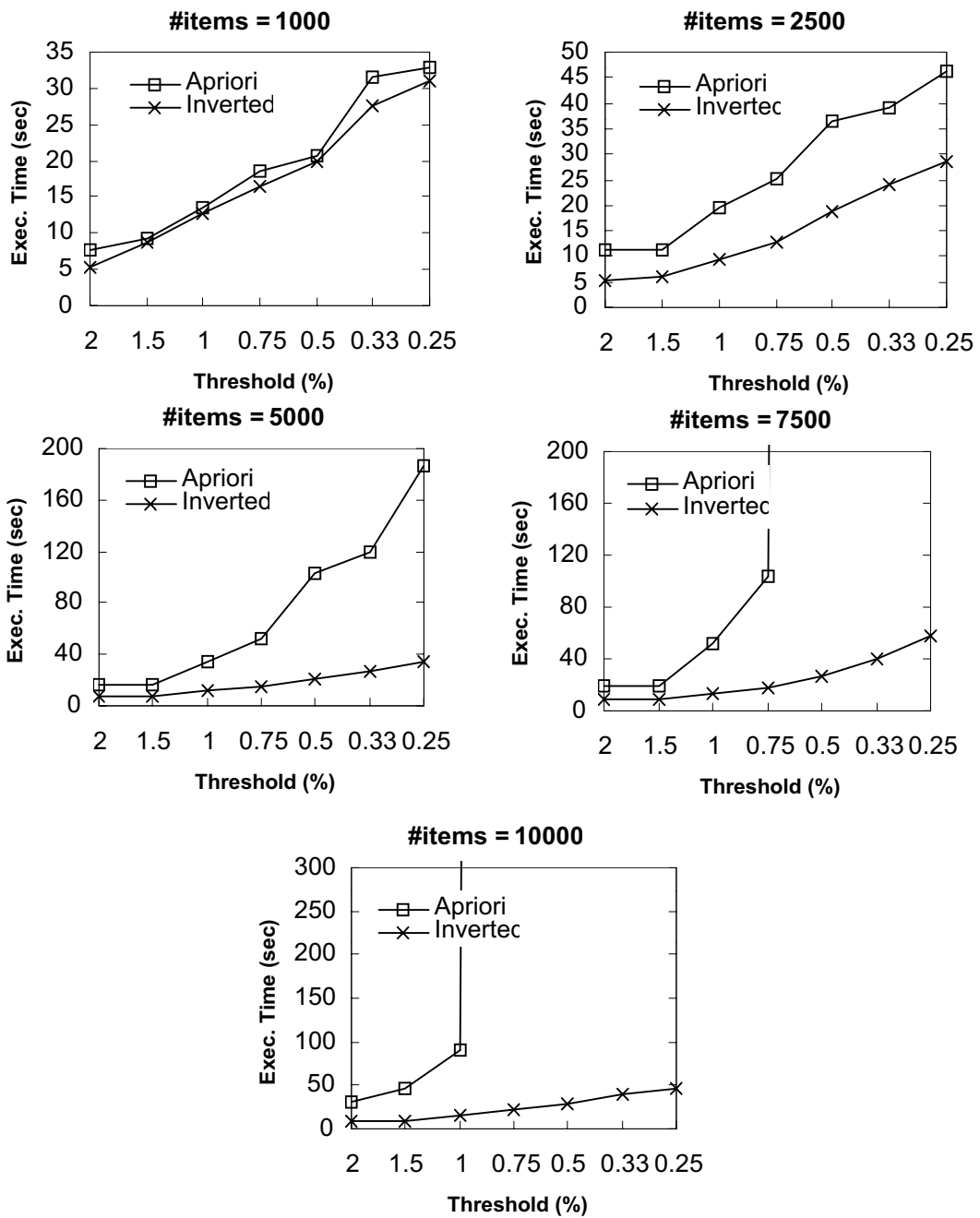


Figure 6. Problem size scale-up