AN ANALYSIS OF ASSOCIATION RULE MINING ALGORITHMS

Renáta Iváncsy
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
renata.ivancsy@aut.bme.hu

Ferenc Kovács
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
kovacsf@aut.bme.hu

István Vajk
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
vajk@aut.bme.hu

ABSTRACT

The association rule mining is a fundamentally important task in the process of knowledge discovery in large databases. Several algorithms have been developed for single-level, single-dimensional, Boolean association rule mining. Some of them require a small amount of memory, but heavy disk access (such as Apriori-like algorithms); others necessitate low I/O activity, but large amount of memory (such as FP-growth). Different algorithms support different applications and requirements depending on the technical background. For this reason it is desirable to classify these algorithms.

In this paper a trade-off is illustrated, namely, which aspects of selection should be considered, when one classifies association rule mining algorithms. Well known algorithms are categorized with these criteria, and the concept of restricted association rule mining is introduced. Necessary modifications are also shown to the algorithms assuming that not all frequent itemsets are needed, only those with maximal size of a given threshold. The paper examines the mining time for both the original and the modified algorithms, and calculates the profit.

1. INTRODUCTION

The association rule mining is a fundamentally important task in the field of data mining. It is a process of discovering not trivial relationships between data in large databases. The problem of association rule mining was first introduced by Agrawal et al in 1993 [1]. Since then it is one of the most popular research area on the field of knowledge discovery.

The association rule mining problem is commonly known as the market basket analysis, but there are several applications that use association rules as well i.e. biological research areas, telecommunication and network analysis etc.

Regarding the diversity of the applications that use association rule mining, several algorithms have been developed. All of these algorithms have their own advantages and disadvantages, so it is useful to compare them. Most of the algorithms find all frequent itemsets but there are several applications that do not need all of them. For this purpose we introduce the restricted itemset mining problem in which the algorithm finds only the maximal k-frequent itemsets.

The organization of the paper is as follows. Section 2 introduces the problem of association rule mining. Section 3 discusses the aspects of classifying the frequent itemsets mining algorithms. In Section 4 we classify the most common known algorithms based on these aspects. In Section 5 we introduce the restricted sized itemset mining problem which means, that the algorithm need to discover not all frequent itemsets, but only those, which size is less than a given threshold. We also show in this section, how the algorithms have to be modified for this purpose. In section 5 we present an experimental analysis of the original and the modified algorithms. We conclude in section 7.

2. PROBLEM STATEMENT

The association rule mining problem is defined as follows. Let $I = \{i_1, i_2, ...i_n\}$ be the set of items. Let D be the set of task relevant data. D contains database transactions T_i, where transaction T_i is a subset of the items in I. The transaction T_i also contains an identifier *TID*. Let A and B be a set of items. Then the association rule is defined as follows: $A \xrightarrow{s,c} B, \text{where } A \subset I, B \subset I, \text{and } A \cap B = \varnothing$. We say, that the rule $A \xrightarrow{s,c} B$ holds in the transaction set D with support s and confidence c, where s is the percentage of transactions in D that contain both A and B, and c is the percentage of transactions in D containing A that also contain B. There are two thresholds for these two values called minimum support (minsup) and minimum confidence (minconf) threshold. Rules satisfying both are called strong association rules.

Those itemsets, which support is over the minimum support threshold are called frequent itemsets (FI). The itemset X is called a maximal frequent itemset (MFI), if X is frequent, and there is no superset of X, that is frequent. The itemset X is called frequent closed itemset (FCI), if there does not exist an

itemset *X'* such that *X'* contains *X* and t(*X*) = t(*X'*), where t(*X*) denotes the set of transactions containing the itemset *X*.

The process of the association rule mining can be divided into two separate phases. In the first step the frequent itemsets have to be discovered. In the second step the association rules have to be generated from these frequent itemsets. The frequent itemsets are needed not only to generate association rules, but discovering frequent itemsets is also the first step of other data mining task i.e. sequential pattern mining, etc.

The frequent itemsets are determined from the original database, which can scale up to terabytes in size. The association rules are retrieved from the frequent itemsets, which size is much smaller than the size of the original database. It is conceivable that from the two steps the first one is the most time consuming [1], so most of the published association rule mining algorithms deal with the problem of efficiently discovering the frequent itemsets.

Certain applications deal with data of various characteristics. In one case, the size of the transactions can be relatively short, for example in market basket data; in other cases it can be relatively long, for example in biological data. So the length of the frequent pattern can also be short or long. In the first case, we say that the database is spare in the second case we say it is dense.

3. ASPECTS OF CLASSIFYING

In this paper we classify the frequent itemset mining algorithms considering the following aspects:

- The type of the discovered frequent itemset
- Using candidates
- The representation of the transactions
- The itemsets representation used in the algorithm
- The number of disk access
- The completeness of the generated frequent itemsets
- The length of the maximal frequent pattern

The type of the discovered frequent itemset

There is an essential difference what the algorithm searches for. An algorithm may discover all frequent itemsets, or the frequent closed itemsets or only the maximal frequent itemsets.

By means of the frequent itemsets all the association rules can be determined. The number of these rules is huge and more importantly some of them are redundant. Therefore the concept of frequent closed itemsets and the concept of association rules based on frequent closed itemsets were introduced in [2]. It is susceptible of proof that all association rules that can be deducted from the frequent itemsets can also be deducted from the association rules based on frequent closed itemsets. Because the cardinality of the frequent closed itemsets is much smaller, than the cardinality of the frequent itemsets, many algorithms solve the problem of finding the frequent closed itemset in place of finding all frequent itemsets. These algorithms are commonly faster.

Mining the maximal frequent itemset lead to a loss of information, because it does not contain the support information of the subsets and the maximal frequent itemsets only in themselves are not useful for generating association rules. There are applications where the set of maximal patterns is needed, such as combinatorial pattern discovery in biological applications [3]. In those cases the purpose is to determine the maximal frequent itemsets, because the set of MFI is in orders of magnitude smaller than the set FCI, and FCI is in orders of magnitude smaller than FI. So it is easy to see, that MFI $\subseteq$ FCI $\subseteq$ FI.

Using candidates

The algorithms can be distinguished whether it uses candidates during the mining process, or not. This can be important, because in many cases the number of candidates can be very large, while the number of the frequent itemsets is small. In this case using candidates may have a drawback.

Certain algorithms generate candidates during the discovering process of the frequent itemsets. These algorithms generate first candidates, than each single candidate is checked, if its support is greater than the minimum support threshold. Other algorithms don't use candidates to discover the frequent itemsets. Both of the two algorithms have its advantage and disadvantage. The disadvantage of using candidates is in general, that there are a huge amount of candidates, and only a few are proved frequent.

The representation of the transactions

It is of interest, and it is considerable which transaction-representation should be used by the algorithm. The decision on choosing the right transaction representation depends on several factors. One of them is how the original dataset or database is organized that is, in which form is the data available for the algorithm. Another point of view is which representation is more suitable for the internal operation of the specific algorithm. The representation of the transactions can be fundamentally the following: horizontal item list, horizontal item vector, vertical TID (transaction identifier) list, and vertical TID vector.

Each representation has its advantages and disadvantages. It is worth considering which form of data the actual algorithm should deal with. There are two ways organizing the transactions. In the one way for each transaction the items are enumerated, that are held by the given transaction. In the other way for each item the transactions IDs are enumerated, that holds the given item. There are four groups of transaction representation.

- *Horizontal item list (HIL)*
  The representation of a transaction contains a transaction id (TID), and for each transaction, there is an ordered list of the items.
- *Horizontal item vector (HIV)*
  The representation of a transaction contains the TID, and the items belonging to a transaction is represented in a bit vector. The length of the vector is the same as the number of the items (n). The i-th bit in the vector is 1, if the transaction holds the i-th item otherwise it is 0.
- *Vertical TID list (VTIDL)*
  For each item there is a list of TIDs. These TIDs are the IDs of those transactions that hold the given item.
- *Vertical TID vector (VTIDV)*
  In this case, for each item there is a bit vector. The length of the vector equals the number of the transactions in the database. For each transaction belongs a bit. If the i-th transaction holds the given item, than the i-th bit is 1 in the vector, otherwise 0.

The advantage of the list representation is, that we use storage space only for those items (or transactions), which belong to the given transaction (or item). In the case, if the number of the items in a transaction is relatively small, it is worth using this representation. There is a case for the horizontal item list that the data is available almost in this form; it is no need for transformation from the one representation to another. The advantage of the vertical TID vector is, that we can calculate the support of an itemset only by intersecting (logical AND operation) the two vectors, that belong to the parts of the itemset unlike in the case of horizontal representation, when we have to go through all the transactions list, and count the occurrences of the given items. The vertical TID vector can be more efficient than the vertical TID list, if the average number of the occurrence is bigger than N/32 (where N denotes the number of transaction in the database), because if we store a TID in 32 bits, and an item occurs in more than N/32 transactions, then it is more efficient to hold them in a vector [3]. It may happen, that we use various form of representations during the algorithm depending on which form fits better to our needs.

The itemsets representation used in the algorithm

Regarding the computational costs and the performance it is an important aspect, how the algorithms handles the data. A hash-tree, a lexicographical-tree or a bipartite graph can be built. Another important aspect is how the tree is traversed.

The itemsets representation used in the algorithm can be diverse. The most common used structure is the lexicographical tree. The lexicographical tree is defined as following [4]:

(1) To each frequent itemset belongs a node in the tree. The root node belongs to the null itemset.
(2) Let $I=\{i_1,\ldots i_k\}$ be an itemset, where i_1, i_2, .. i_k are listed in lexicographical order. The parent of the node I is the itemset $\{i_1,\ldots i_{k-1}\}$

The algorithms that use this structure can differ in the way, how the tree is traversed. It can be a depth-first strategy (DFS), a breadth-first strategy (BFS), or a hybrid strategy. The algorithms reduce the searching area by pruning the tree based on a priori information.

In another case, not the candidate itemsets are represented during the algorithm, but the transactions itself. Here a tree contains the transactions in a compressed way, and the frequent itemsets can be discovered from this tree without candidate generation. For example the FP-growth algorithm [5] that transforms the database into an FP-tree.

The third form is to use a bipartite graph. Bipartite graphs are graphs, in which we can partition the vertices in two sets A and B. The vertices, that belongs to A can be connected only with vertices that belong to B and vice versa. The one set of the vertices are the transactions, the other set's vertices are the items. There is an edge between a transaction and an item, if the transaction holds the item. By using bipartite graphs, the problem of finding the frequent itemsets is finding maximal cliques in the graph. This problem is NP-complete.

The number of disk access

Regarding the I/O cost, it is a very essential parameter, how many times the algorithm accesses the database. We distinguish level wise algorithms, two-phase algorithms (database compression methods or sampling) and partitioning algorithms.

The number of disk access is critical in data mining task, because the I/O operation is more time consuming than a memory operation. It can be of advantage when we minimize the disk access. Regarding the database access, we encounter two main classes of the algorithms: level wise algorithms and two phase algorithms. The level wise algorithms read the whole database k times for finding the k-frequent itemset. The algorithm will read the database as many times, as long the longest itemset is. If there is only one long frequent itemset, the mining time will increase rapidly. There are several ideas how the mining time can be reduced for example with hash based algorithms [6], dynamic itemset count methods [7] and so on, but using these methods does not speed up the mining significantly.

The two-phase mining algorithms read the database only twice. There are several cases to accomplish that. In one case by the first database reading the support count of the 1-frequent itemsets are determined, and using this information reading the second time the database we can build the whole database in the memory (with a clever compression of the database for example building a tree [5]). The mining operation has to be accomplished only in the memory.

The second method is to divide the database into small parts, so that one part fit into the memory. Afterwards we can use a mining algorithm with a decreased minimum support threshold on this small database in the memory. In this case, after the first database read we have locally frequent itemsets that can be greater than the frequent itemsets in the original database, because it can happen, that an itemset is frequent locally, but not frequent globally. So a second database reading is needed to determine for each locally frequent itemset whether it is also frequent globally.

The third method is the sampling. In this method, in the first step the database is sampled so that the sampled database fits into the main memory. In the second step the mining algorithm is executed in the memory. The third step and the second database scan is the verifying, checking of the frequent itemsets if they are really frequent.

The completeness of the generated frequent itemsets

By the sampling algorithms it can happen, that the algorithm does not find all frequent itemsets. It is possible, that a given application does not need it, so these algorithms have also justification.By the sampling algorithm it can happen that the statistical features of the sampled database do not match with the statistical features of the original database. In this case not all frequent itemsets will be found. Several algorithms ignore this, but there are algorithms which can indicate whether all the frequent itemsets were found or not [8].

The length of the maximal frequent pattern

The databases belonging to the various applications have various characteristic. One fundamental difference regarding the algorithm is whether the database is spare or dense, namely how long patterns contain the transactions.

The algorithm's performance depends on the features of the patterns in the database. There is a significant difference between the methods finding short and long frequent itemsets. The long patterns can be found by the depth-first traversing of the lexicographical tree, while the short patterns are found

quickly by using breadth-first traversing in a lexicographical tree or by using level wise algorithms.

4. CLASSIFICATION OF THE ALGORITHMS

The algorithms can be classified differently regarding the aspects discussed in the previous section. In our classification the aspects of using candidates and of the type of the frequent patterns have the highest priority. We have chosen these two aspects because they differentiate the algorithms significantly. It is hard to compare –as well regarding the mining time, as well regarding the memory usage – two algorithms, if one of them find all frequent itemsets, and the other only the maximal frequent itemsets. The Figure 1 differentiates the known algorithms which can be found in the references regarding these aspects into five categories.

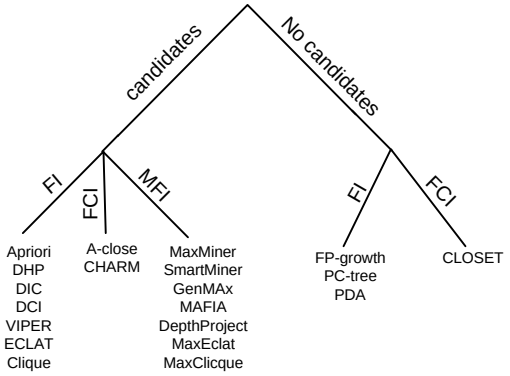

Figure 1. The classification of the algorithms

The main features of the referenced algorithms regarding the mentioned aspects are shown in Table 1:

	Algorithm	Representation of the transactions				Used structure			
		HIL	HIV	VTIDL	VTIDV	BFS	DFS	FP-tree	Bipartite Graph
FI	Apriori [9]	✓				✓			
	DHP[6]	✓				✓			
	DIC [7]	✓				✓	✓		
	DCI [10]	✓		✓					
	VIPER [11]				✓				
	ECLAT [12]			✓			✓		
	Clicque [12]			✓					✓
	FP-growth [5]	✓						✓	
	PC-tree [13]	✓						✓	
FCI	A-Close [2]	✓				✓			
	CLOSET [14]	✓						✓	
	CHARM [15]	✓				✓			
MFI	MaxMiner [16]	✓				✓			
	SmartMiner [17]				✓		✓		
	GenMax [18]			✓		✓			
	MAFIA [3]				✓		✓		
	DepthProject [19]				✓		✓		
	MaxEclat [12]			✓			✓		
	MaxClicque [12]			✓					✓

Table 1. The comparison of the algorithms

5. SIZE RESTRICTED FREQUENT ITEMSETS

To meet the needs of the end user it is important that the information discovered in the large database will be useful for the user. It is not profitable when the information can be processed by an expert only, who understands data mining in detail. To achieve easy-to-use services there are several methods. The first one is to discover only the frequent closed itemsets that is in order of magnitude smaller than the set frequent itemsets. From the set of frequent closed itemsets all association rules can be discovered. Unfortunately it possibly occurs that the number of the frequent closed itemset can also be too huge for human processing. Another key issue is that after the data mining process we use a post-processing algorithm for understanding the mined rules. This is almost necessary, but it makes difference what amount of information is to be post-processed.

It may occur that the user establishes a claim to mine only the k-frequent itemsets. It is not presumptive that a manager in a supermarket will see frequent itemsets that are longer than 3 or 4 items. If we modify the algorithms only to discover the maximal k-frequent itemsets we can save time.

The easiest way to mine only the maximal k-frequent itemset is to modify the level wise algorithms (Apriori, DHP, DIC and so on). In this case we have to stop the algorithm after it founds the k-frequent itemsets. Similarly by the lexicographical tree traversing algorithms we have to limit the traversing of the tree by the k-th level independent from the traversing method (breadth-first or depth-first).

The FP-growth algorithm first reads the database and counts the occurrences of the items. Regarding the 1-frequent items by the second database read it builds an FP-tree in the memory. After that it recursively generates conditional FP-trees and generates the frequent patterns without generating candidates. If we want to restrict the size of the frequent itemsets to be discovered, we have to make a modification in two places in the algorithm. We have to restrict the level of the recursion and we have to generate only the subsets of the single path tree with size of the threshold. The PC-tree algorithm does not differ significantly from the FP-growth, so the way to restrict it is the same. The CLOSET is a modification of the FP-growth that finds only the frequent closed itemsets. It makes no sense to restrict the algorithms that discover only the maximal frequent itemsets because its aim is to find the long patterns in the database.

6. SIMULATION

To examine the speed-up of the algorithms when restricting the itemset size to be mined we implemented two fundamental algorithms, the Apriori and the FP-growth algorithm. The first one is a level wise algorithm, the second one is an algorithm, that does not use candidates, but it needs a huge amount of memory. The algorithms were implemented in C#. The simulations were executed on a Pentium 4 CPU, 2.40 GHz, and 512MB of RAM computer.

The datasets used by the algorithms are semantic datasets from [9]. We used two datasets for the experimentation. The T20I5D10K dataset means that there are 10K transactions in it, the average size of a transaction is 20, and the average size of the maximal frequent itemset is 5. Similarly the T20I7D10K means 10K transactions, the average size of a transaction is 20 and the average size of the maximal frequent pattern is 7.

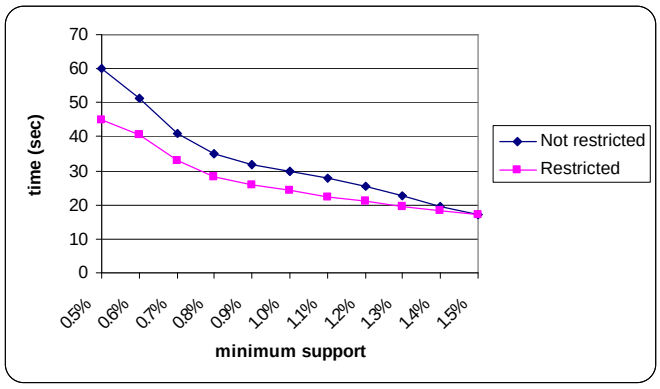

Figure 2. The execution time of the Apriori algorithm, T20I5D10K

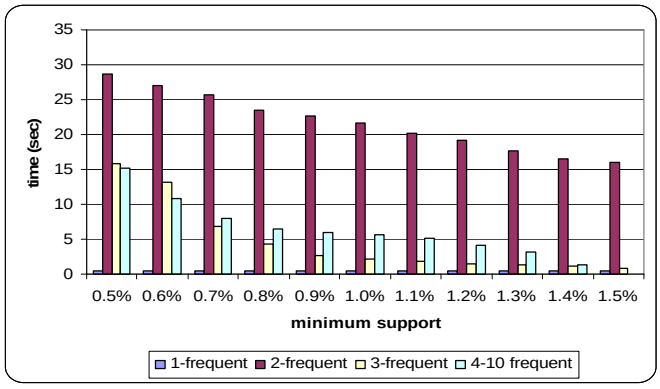

Figure 3. The time to be saved when restricting the itemset size to 3, T20I5D10K

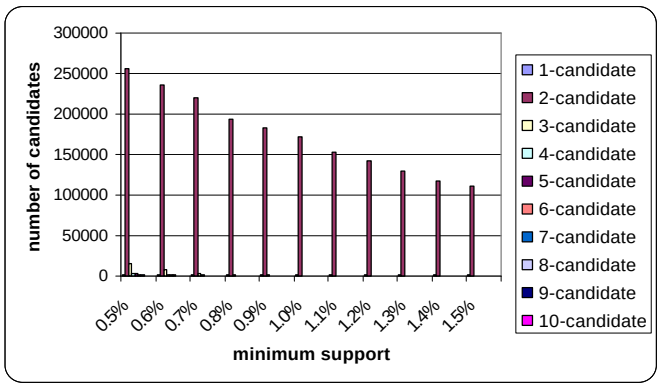

Figure 4. The number of candidates, T20I5D10K

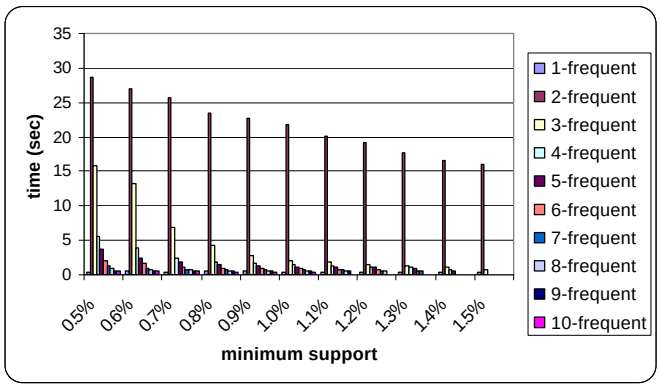

Figure 5. The execution time per itemset levels, T20I5D10K

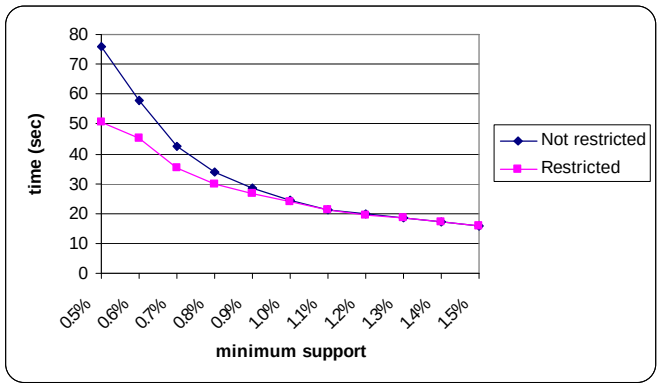

Figure 6. The execution time of the Apriori algorithm, T20I7D10K

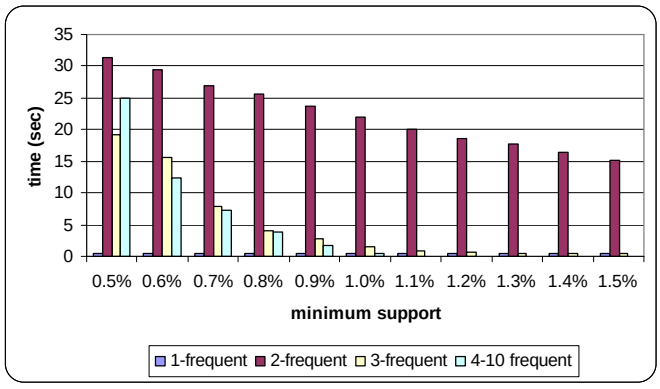

Figure 7. The time to be saved when restricting the itemset size to 3, T20I7D10K

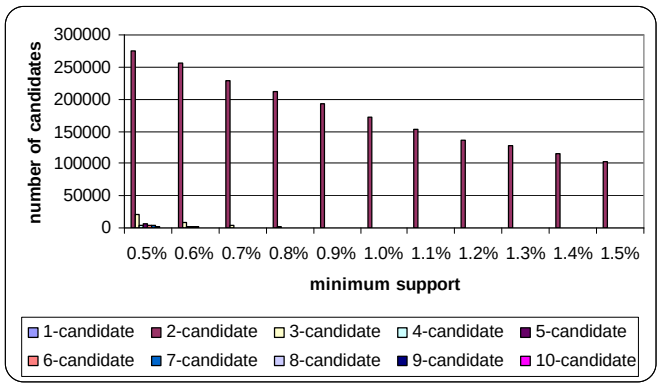

Figure 8. The number of candidates, T20I7D10K

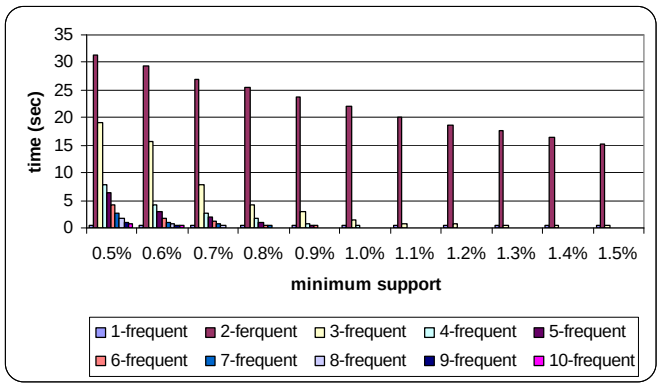

Figure 9. The execution time per itemset levels, T20I7D10K

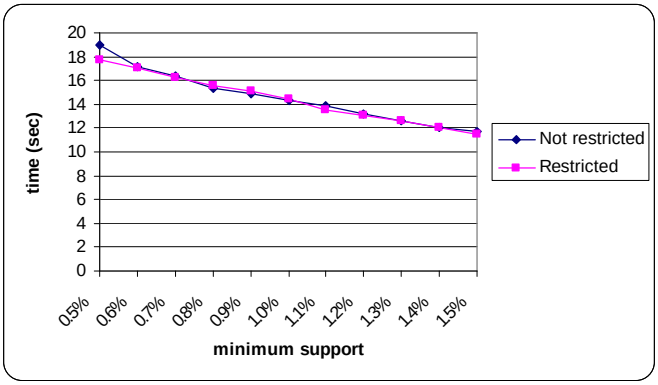

Figure 10. The execution time of the FP-growth algorithm, T20I5D10K

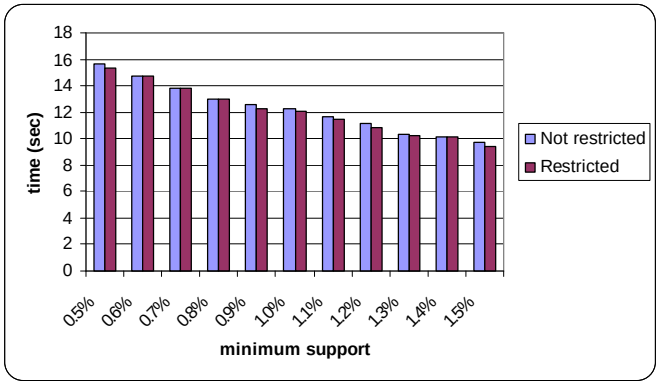

Figure 11. The recursive pattern growth time, T20I5D10K

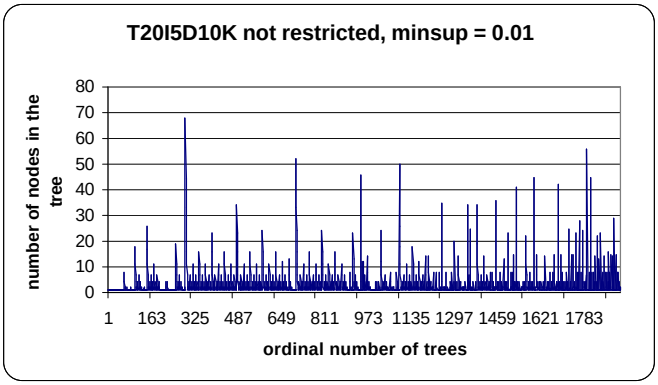

Figure 12. The size of the created trees during the pattern growth phase in not restricted mining

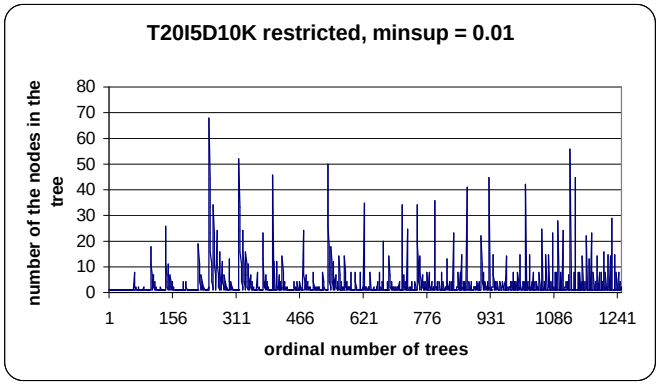

Figure 13. The size of the created trees during the pattern growth phase in restricted mining

Figure 1 shows two executions times of the Apriori algorithm on the dataset T20I5D10K as a function of the minimum support. The difference between the two executions is that the first was accomplished without restriction, and the second was restricted to 3. It is fully visible, that the restricted algorithm is faster, especially by lower minimal support threshold. In Figure 3 there are four times represented for each minimum support. The first three are the execution time for mining the 1-frequent, the 2-frequent and the 3-frequent itemsets. The fourth time is for the rest of the algorithm. It shows, how many time can be saved restricting the itemset size to 3. One can see that from 1% down to 0.5% the time saving keeps increasing. Figure 4 presents the sizes of the candidates in each level. In Figure 5 are the times depicted for processing the candidates in each level and finding the frequent itemsets. We can draw a conclusion that the Apriori algorithm spends most of its time generating and checking the 2-candidates.

Figure 6, 7, 8, and 9 represents the same as the figures before but for the dataset T20I7D10K. We can see that the time gain by lower support is longer than that of dataset T20I5D10K.

Figure 10 shows the execution times for the original and the restricted FP-growth algorithms as a function of the minimum support. It can be seen that in the restricted case the execution time is not significantly smaller than in case of the original algorithm in contrary to our expectations. For the explanation let us divide the execution time of the FP-grow algorithm in the following three parts:

(1) Time for counting the 1-frequent items
(2) Time for building the first tree
(3) Time for the recursive pattern growth process that includes the construction of further trees

Considering the way for restriction outlined in the previous section we can realize that the times for the first two parts are the same in the original and in the modified algorithms. The difference is only in the third part that is the recursive pattern growth time. Figure 11 represents the recursive pattern growth time for both the original and the restricted algorithm. We can see that there is hardly any difference between the two cases. The reason for this is depicted in Figure 12 and Figure 13. Figure 12 illustrates the size of the trees (that is the number of nodes in the tree) built during the recursive pattern growth part of the original algorithm. The first tree is not shown because the building time for it is the same in both cases and its size is significant higher (162190 nodes in this case) than the size of the further trees. Figure 13 illustrates the size of trees of the restricted algorithm. Apparently the restricted algorithm generates fewer trees in the pattern growth phase than the original algorithm does. Comparing the two figures it is obvious that the two algorithms differs only in the number of the small trees that is the restricted algorithm generates all the big trees that the original algorithm generates, the saving is rather by the trees having between 1 and 20 nodes.

Apparently, restricting the Apriori algorithm results in significant time savings especially at lower minimum support, but restricting the FP-growth algorithm produces no significant time saving.

7. CONCLUSION

In this paper we have covered some of the efficiency issues of the algorithmic association rule mining. The aim of our examination was to classify the most commonly known frequent itemset discovering algorithms. We have established a system of aspects for classifying these algorithms. We divided the algorithms in two main classes namely the class of algorithms that use candidates and of those that do not. Within these two classes we defined another three classes based on the types of the discovered frequent itemsets (FI, FCI, and MFI).

In the second part of the paper we have introduced a new category, the restricted itemset. This means that the algorithm does not have to discover all frequent itemsets only those have smaller size than a given threshold. We presented how the two of the main frequent itemset discovering algorithms – the Apriori and the FP-growth – should be modified for this purpose. Experimental results show that the algorithms are faster restricting the itemset size. In the case of the Apriori algorithm, the time saving in lower support threshold is significant, but in the case of FP-growth, the gain is very small. This phenomenon has been explained by the size and number of the trees generated by the recursive pattern growth part of the algorithm.

ACKNOWLEDGMENTS

This work has been supported by the fund of the Hungarian Academy of Sciences for control research and the Hungarian National Research Fund (grant number: T042741)

REFERENCES

[1] R. Agrawal, T. Imielinski and A.Swami: „Mining association rules between sets of items of large databases". *Proc. of the ACM SIGMOD Intl'l Conf. On Management of Data, May 1993.*

[2] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. "Discovering frequent closed itemsets for association rules" *In Proc. of Intl. Conference on Database Theory (ICDT), January 1999.*

[3] D. Burdick, M. Calimlim, and J. Gehrke: "MAFIA: a maximal frequent itemset algorithm for transactional databases" *In Intl. Conf. on Data Engineering, Apr. 2001.*

[4] R. Agarwal, C. Aggarwal, and V. Prasad. "A tree projection algorithm for generation of frequent itemsets." *In Proceedings of the High Performance Data Mining Workshop*, Puerto Rico, 1999.

[5] J.Han, J. Pei and Y. Yin: „Mining frequent patterns without candidate generation" *In Proc. of the 2000 ACM-SIGMOD Int'l Conf. On Management of Data, Dallas, Texas, USA, May* 2000

[6] J. S. Park, M. Chen, and P. S. Yu: "An effective hash based algorithm for mining association rules" *Proc. of the 1995 ACM Int. Conf. on Management of Data, San Jose, California, 1995*

[7] S. Brin, R. Motwani, J. D. Ullman and S. Tsur: „Dynamic itemset counting and implication rules for market basket data" *SIGMOD Record, 1997*

[8] H. Toivonen: "Sampling large databases for association rules" *In the VLDB Journal, pages 134-156, 1996*

[9] R. Agrawal and R. Srikant: „Fast algorithms for mining association rules", *Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. 1994*

[10] S. Orlando: "High performance mining of short and long patterns" *2001.*

[11] P. Shenoy, J. Haritsa, G. Bhalotia, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging Vertical Mining of Large Databases" *Proceedings of the ACM SIGMOD, Dallas, TX, May 2000, pp. 22-33.*

[12] M. J. Zaki: "Scalable algorithms for association mining" *IEEE Transaction on Knowledge and Data Engineering. Vol 12. No 3. May/June 2000.*

[13] V. S. Ananthanarayana, D. K. Subramanian and M. N. Murty: "Scalable, distributed and dynamic mining of association rules" *High Performance Computing (HIPC) 2000.*

[14] J. Pei, J. Han, and R. Mao: "CLOSET: An efficient algorithm for mining frequent closed Itemsets". *In Proc. of the 2000 ACM-SIGMOD Int'l Conf. on Management of Data, Dallas, Texas, USA, May 2000.*

[15] M. J. Zaki and C.-J. Hsiao: "CHARM: An efficient algorithm for closed association rule mining." *TR 99-10, CS Dept., RPI, Oct. 1999.*

[16] R. J. Bayardo: "Efficiently mining long patterns from databases" *In ACM SIGMOD Conf., June 1998*.

[17] Q. Zou, W. W. Chu, B. Lu: "SmartMiner: A depth first algorithm guided by tail information for mining maximal frequent itemsets" *IEEE ICDM 2002*.

[18] K. Gouda, M. J. Zaki: "Efficiently mining maximal frequent itemsets" *ICDM'01, San Jose, California, Dec. 2001.*

[19] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad: "Depth first generation of long patterns" *In Proc. Sixth ACM SIGKDD*, 2000.

[20] Qinghua Zou, Wesley Chu, David Johnson, and Henry Chiu "Pattern Decomposition Algorithm for Data Mining Frequent Patterns" *J. Knowledge and Information Systems(KAIS), 2002*