B. Yildiz and B. Ergenc. "Comparison of Two Association Rule Mining Algorithms without Candidate Generation". In the 10th IASTED International Conference on Artificial Intelligence and Applications (AIA 2010), Innsbruck, Austria, pp.450-457, Feb 15-17, 2010

COMPARISON OF TWO ASSOCIATION RULE MINING ALGORITHMS WITHOUT CANDIDATE GENERATION

Barış Yıldız, Belgin Ergenç
Department of Computer Engineering
Izmir Institute of Technology
Gulbahce Koyu, 35430 Urla
Izmir, Turkey
barisyildiz@iyte.edu.tr, belginergenc@iyte.edu.tr

ABSTRACT
Association rule mining techniques play an important role in data mining research where the aim is to find interesting correlations among sets of items in databases. Although the Apriori algorithm of association rule mining is the one that boosted data mining research, it has a bottleneck in its candidate generation phase that requires multiple passes over the source data. FP-Growth and Matrix Apriori are two algorithms that overcome that bottleneck by keeping the frequent itemsets in compact data structures, eliminating the need of candidate generation. To our knowledge, there is no work to compare those two similar algorithms focusing on their performances in different phases of execution. In this study, we compare Matrix Apriori and FP-Growth algorithms. Two case studies analyzing the algorithms are carried out phase by phase using two synthetic datasets generated in order i) to see their performance with datasets having different characteristics, ii) to understand the causes of performance differences in different phases. Our findings are i) performances of algorithms are related to the characteristics of the given dataset and threshold value, ii) Matrix Apriori outperforms FP-Growth in total performance for threshold values below 10%, iii) although building matrix data structure has higher cost, finding itemsets is faster.

KEY WORDS
Data Mining, Association Rule Mining, FP-Growth, Matrix Apriori

1. Introduction

Data mining defined as finding hidden information from large data sources has become a popular way to discover strategic knowledge. Direct mail marketing, web site personalization, bioinformatics, credit card fraud detection and market basket analysis are some examples where data mining techniques are commonly used.

Association rule mining is one of the most important and well researched techniques of data mining. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transactional databases or other data repositories [1]. If you consider market basket data, the purchasing of one product(X) when another product(Y) is purchased represents an association rule [2] and displayed as X→Y. Association rule mining process consists of two steps: finding frequent itemsets and generating rules. The rules are generated from frequent itemsets. An itemset is a set of items in the database. Frequent itemset is an itemset of which support value (percentage of transactions in the database that contain both X and Y) is above the threshold defined as minimum support. The main concentration of most association rule mining algorithms is to find frequent itemsets in an efficient way to reduce the overall cost of the process.

Association rule mining was first introduced by [3], and in [4] the popular Apriori algorithm was proposed. It computes the frequent itemsets in the database through several iterations. Each iteration has two steps: candidate generation and candidate selection [5]. Database is scanned at each iteration. Apriori algorithm uses large itemset property: any subset of a large itemset must be large. Candidate itemsets are generated as supersets of only large itemsets found at previous iteration. This reduces the candidate itemset number. Among many versions of Apriori [6 and 7], FP-Growth has been proposed in association rule mining research with the idea of finding frequent itemsets without candidate generation [8]. FP-Growth uses tree data structure and scans database only twice showing notable impact on the efficiency of itemset generation phase. Lately an approach named Matrix Apriori is introduced with the claim of combining positive properties of Apriori and FP-Growth algorithms [9]. In this approach, database is scanned twice as in the case of FP-Growth and matrix structure used is simpler to maintain. Although it is claimed to perform better than FP-Growth, performance comparison of both algorithms are not shown in that work.

In this study, we analyzed performances of FP-Growth and Matrix Apriori algorithms which are similar in the way of overcoming the bottleneck of candidate generation by the help of compact data structures to find the frequent itemsets. The algorithms are compared not only with their

total runtimes but also with their performances related with the phases. Finding frequent items and building the data structure is analyzed as first phase and finding frequent itemsets is analyzed as second phase. Test runs are carried on two datasets with different characteristics representing the needs of different domains. Impact of the frequent items and the number of frequent itemsets on the performance of algorithms are also observed.
The following of this paper is organized as follows. Section 2 reviews the association rule mining research. Description of the FP-Growth and Matrix Apriori algorithms are given in Section 3, including our implementation steps. In Section 4, phase by phase performance analysis of the evaluated algorithms is given with discussion on the results. Finally, we conclude the paper in Section 5.

2. Related Work

The progress in bar-code and computer technology has made it possible to collect data about sales and store as transactions which is called basket data. This stored data attracted researches to apply data mining to basket data. As a result association rules mining came into prominence which is mentioned as synonymous to market basket analysis. Association rule mining, which was first mention by [3], is one of the most popular data mining approaches. Not only in market business but also in variety of areas association rule mining is used efficiently. In [10], Apriori algorithm is used on a diabetic database and developed application is used to discover social status of diabetics. In a report [11], association rules are listed in the success stories part and in a survey [12] the Apriori algorithm is listed in top 10 data mining algorithms.
The proposed algorithm in [3] makes multiple passes over database. In each pass, beginning from one element itemsets, the support values of itemsets are counted. These itemsets are called candidate itemsets which are extended from the frontier sets delivered from previous pass. If a candidate itemset is measured as frequent then it is added to frontier sets for the next pass.
The Apriori algorithm proposed in [4] boosted data mining research with its simple way of implementation. The algorithm generates candidate itemsets to be counted in a pass by using only the itemsets found large in previous pass – without considering all of the transactions in the database. So too many unnecessary candidate generation and support counting is avoided. Apriori is characterized as a level-wise complete search algorithm using anti-monotonicity of itemsets, "if an itemset is not frequent, any of its superset is never frequent" [12 and 13].
There have been many improvements like [6 and 7] for Apriori algorithm but the most significant one is FP-Growth algorithm proposed in [8]. The main objective is to skip candidate generation and test step which is the bottleneck of the Apriori like methods. The algorithm uses a compact data structure called FP-tree and pattern fragment growth mining method is developed based on this tree. FP-growth algorithm scans database only twice. It uses a divide and conquer strategy. Algorithm relies on Depth First Search scans while in Apriori Breath First Search scan is used [14]. It is stated in [8] that FP-growth is at least an order of magnitude faster than Apriori.
In several extensions for both Apriori and FP-growth accuracy of results is sacrificed for better speed. Matrix Apriori proposed in [9], combines positive properties of these two algorithms. Algorithm employs two simple structures: A matrix of frequent items called MFI and a vector storing the support of candidates called STE. Matrix Apriori consists of three procedures. First builds matrix MFI and populates vector STE. Second modifies matrix MFI to speed up frequent pattern search. Third identifies frequent patterns using matrix MFI and vector STE.
Detailed studies for comparing performances of Apriori and FP-Growth algorithms can be found in [8, 14 and 15]. These studies reveal out that FP-Growth perform better than Apriori when minimum support value is decreased. Matrix Apriori algorithm combining the advantages of Apriori and FP-Growth was proposed as a faster and simpler alternative to these algorithms but there is no work showing its performance. However, a study concentrating on the weaknesses and strengths of Matrix Apriori and FP-Growth would be worth considering since they eliminate the disadvantage of candidate generation of Apriori-like algorithms.

3. Description and Implementation of Algorithms

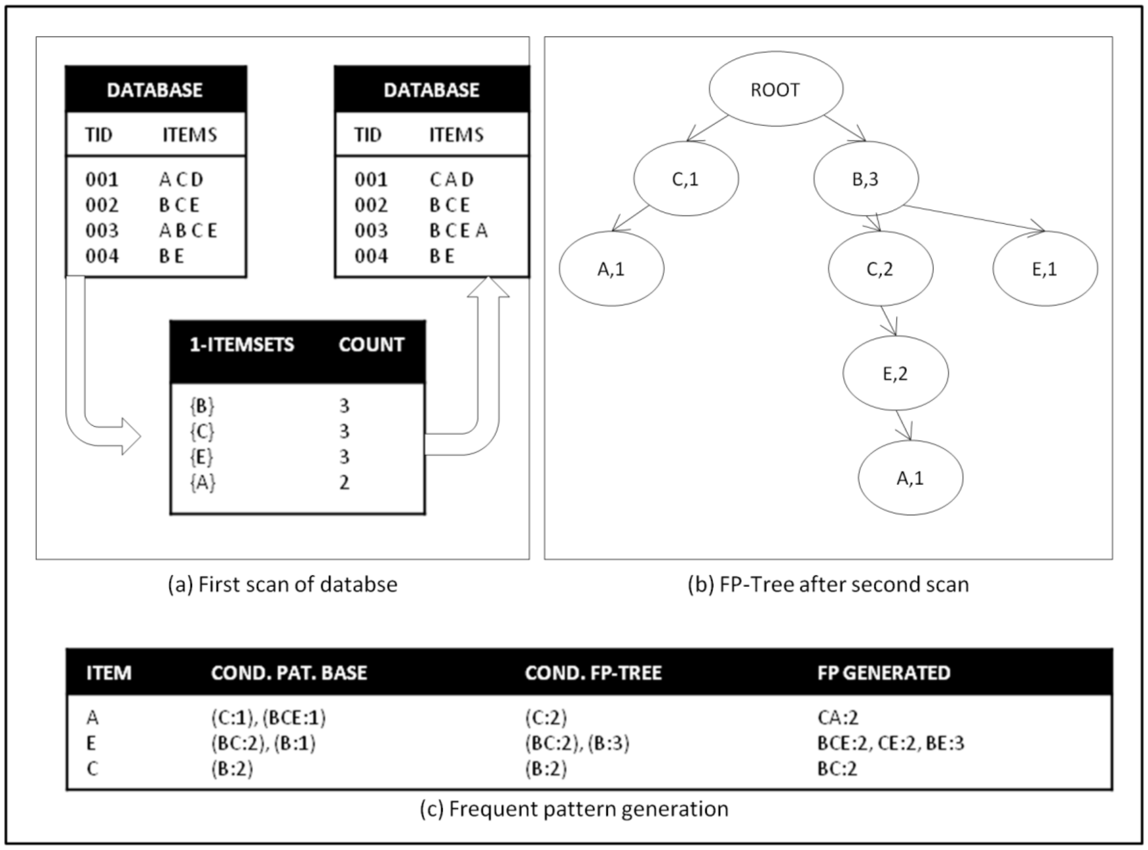
In this part of the paper, firstly FP-Growth and Matrix Apriori algorithms are described to be self contained and following that the implementation steps of the algorithms are given.

3.1 Description

In this section, two association rule mining algorithms are explained: FP-Growth and Matrix Apriori. Both algorithms are proposed as a better alternative to Apriori algorithm in terms of overcoming its bottleneck as several scans of database for candidate generation and testing. FP-Growth was introduced to overcome candidate generation and testing problem. The database is scanned only twice. Matrix Apriori is another approach of which database scan strategy is similar to FP-Growth. Although the scan strategy is similar, the data structures to keep itemsets are different. In following two subsections, the approaches of the algorithms are explained with an example. After description section, in implementation section the steps we go through while implementing the algorithms are given.

3.1.1 FP-Growth

The FP-Growth methods adopts a divide and conquer strategy as follows: compress the database representing frequent items into a frequent-pattern tree, but retain the itemset association information, and then divide such a compressed database into a set of condition databases, each associated with one frequent item, and mine each such database [8].
In Figure 1 FP-Growth algorithm is visualized for an example database with minimum support value 2 (50%). First, a scan of database derives a list of frequent items in descending order (see Figure 1a). Then FP-tree is constructed as follows. Create the root of the tree and scan the database second time. The items in each transaction are processed in the order of frequent items list and a branch is created for each transaction. When considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1. In Figure 1b, we can see the transactions and the tree constructed.



DATABASE

TID	ITEMS
001	A C D
002	B C E
003	A B C E
004	B E

1-ITEMSETS	COUNT
{B}	3
{C}	3
{E}	3
{A}	2

DATABASE

TID	ITEMS
001	C A D
002	B C E
003	B C E A
004	B E

(a) First scan of databse

(b) FP-Tree after second scan

ITEM	COND. PAT. BASE	COND. FP-TREE	FP GENERATED
A	(C:1), (BCE:1)	(C:2)	CA:2
E	(BC:2), (B:1)	(BC:2), (B:3)	BCE:2, CE:2, BE:3
C	(B:2)	(B:2)	BC:2

(c) Frequent pattern generation

Figure 1. FP-Growth example

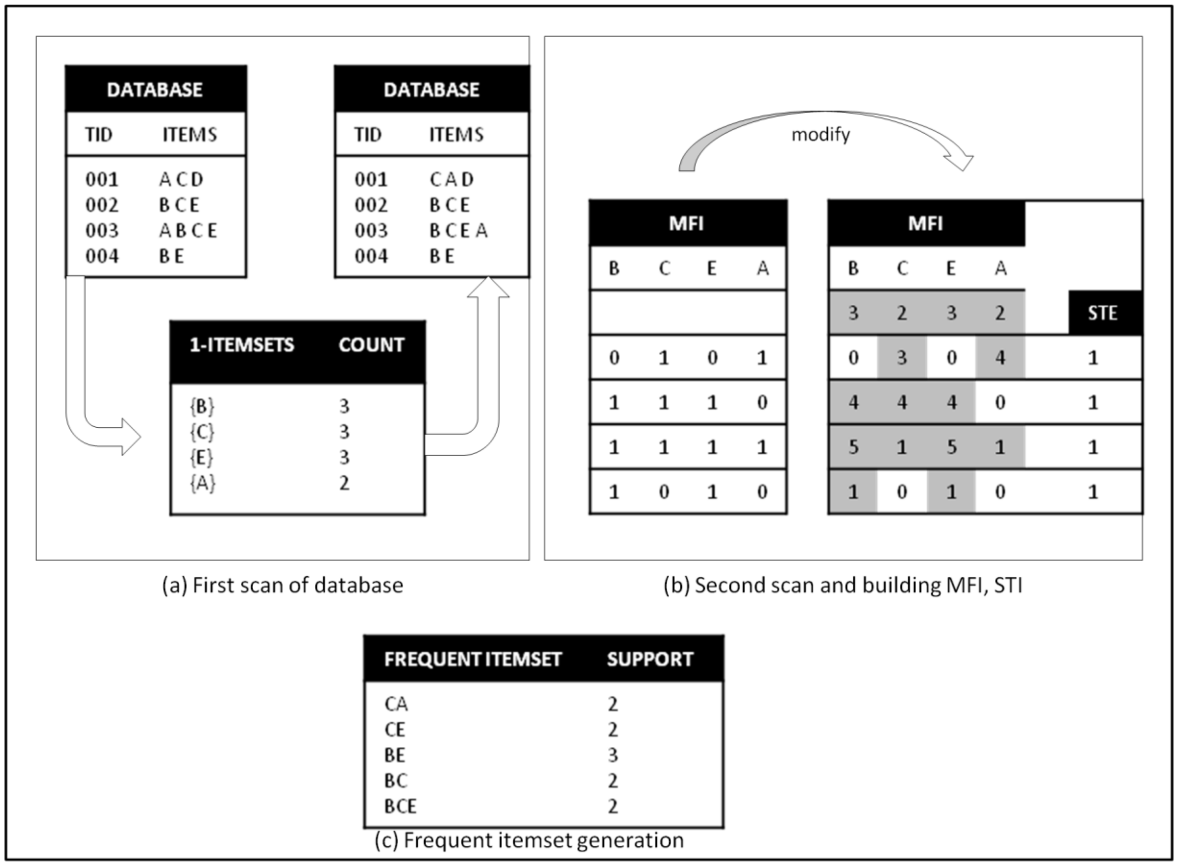
After constructing the tree the mining proceeds as follows. Start from each frequent length-1 pattern, construct its conditional pattern base, then construct its conditional FP-tree and perform mining recursively on such a tree. The support of a candidate (conditional) itemset is counted traversing the tree. The sum of count values at least frequent item's nodes gives the support value. The frequent pattern generation process is demonstrated in Figure 1c.

3.1.2 Matrix Apriori

Matrix Apriori [9] is similar to FP-Growth in the database scan step. However, the data structure build for Matrix Apriori is a matrix representing frequent items (MFI) and a vector holding support of candidates (STE). The search for frequent patterns is executed on this two structures, which are easier to build and use compared to FP-tree.

In Figure 2, Matrix Apriori algorithm is demonstrated. The example database is the same database used in previous section and minimum support value is again 2 (%50). Firstly, a database scan to determine frequent items is executed and a frequent items list is obtained. The list is in descending order (see Figure 2a). Following this, a second scan on database is executed. During the scan the MFI and STE is built as follows. Each transaction is read. If the transaction has any item that is in the frequent item list then it is represented as "1" and otherwise "0". This pattern is added as a row to MFI matrix and its occurrence is set to 1 in STE vector. While reading remaining transactions if the transaction is already included in MFI then in STE its occurrence is incremented. Otherwise it is added to MFI and its occurrence in STE is set to 1. After reading transactions the MFI matrix is modified to speed up frequent pattern search. For each column of MFI, beginning from the first row, the value of a cell is set to the row number in which the item is “1”. If there is not any “1” in remaining rows then the value of the cell is set to “1” which means down to the bottom of the matrix, no row contains this item (see Figure 2b).

After constructing the MFI matrix finding patterns is simple. Beginning from the least frequent item, create candidate itemsets and count its support value. The support value of an itemset is the sum of the items at STE of which index are rows where all the items of the candidate itemset are included in MFI’s related row. Frequent itemsets found can be seen in Figure 2c.



DATABASE	
TID	ITEMS
001	A C D
002	B C E
003	A B C E
004	B E

1-ITEMSETS	COUNT
{B}	3
{C}	3
{E}	3
{A}	2

DATABASE	
TID	ITEMS
001	C A D
002	B C E
003	B C E A
004	B E

(a) First scan of database

MFI			
B	C	E	A
0	1	0	1
1	1	1	0
1	1	1	1
1	0	1	0

MFI				
B	C	E	A	
3	2	3	2	STE
0	3	0	4	1
4	4	4	0	1
5	1	5	1	1
1	0	1	0	1

(b) Second scan and building MFI, STI

FREQUENT ITEMSET	SUPPORT
CA	2
CE	2
BE	3
BC	2
BCE	2

(c) Frequent itemset generation

Figure 2. Matrix Apriori example

It will be beneficial to give a short comparison of given algorithms with an example to show the execution of the algorithms. First scans of both algorithms are carried out in the same way. Frequent items are found and listed in order. During second scan, FP-Growth adds transactions to tree structure and Matrix Apriori to matrix structure. Addition of a transaction to the tree structure needs less control compared to matrix structure. For example, consider 2nd and 3rd transactions. Second transaction is added as a branch to the tree and as a row to the matrix. But addition of third transaction shows the difference. For tree structure we need to control only the branch that has the same prefix with our transaction. So addition of a new branch to node E is enough. On the other hand, for the matrix structure we need to control all the items of rows. If we find the same pattern then we increase the related item of STE. Otherwise we need to scan matrix till we find the same pattern. If we cannot find then a new row is added to matrix. It seems that building matrix needs more control and time, however, management of matrix structure is easier compared to tree structure.

Finding patterns for both algorithms need producing candidate itemsets and control. This is called conditional pattern base in FP-Growth and there is no specific name for Matrix Apriori. Counting support value is easy to handle in Matrix Apriori. However, in FP-Growth traversing the tree is complex.

3.2 Implementation

In this section, we give brief information about the implementation of algorithms. Algorithms, which are explained in previous section, are coded as they are understood from related papers [8 and 9]. For both algorithms the dataset file is read in order to take the information about number of transactions, number of items and name of items to create a temporary file and data mining process is carried out on this file. In this paper, “database” term is used for the temporary file.

3.2.1 FP-Growth

The implementation of FP-Growth is divided into three steps.

Step 1: Database is read and the count of items is found. According to the minimum support threshold, frequent items are selected and sorted.

Step 2: Initialization of the FP-tree is done. From the frequent items a node list is created which will be connected to nodes of the tree. After initialization the database is read again. This time, if an item in a transaction is selected as frequent then it is added to the tree structure.

Step 3: Beginning from the least frequent item, a frequent pattern finder procedure is called recursively. The support count of the patterns are found and displayed if they are frequent.

3.2.2 Matrix Apriori

The implementation of Matrix Apriori is divided into four steps.

Step 1: This is carried out in the same way as step 1 of FP-growth.

Step 2: Initialization of MFI is done. According to frequent items list first row of MFI is created. After initialization, database is read again. Each transaction is converted to an array. This array is at length of MFI's one row. If in MFI there is a pattern same as the array of transaction then its occurrence is increased. Otherwise a new row is added to MFI.

Step 3: MFI is modified. This modification will speed up the pattern search and support counting process.

Step 4: Similar to FP-Growth beginning from the least frequent item a procedure is called recursively and support values of patterns are counted.

The implementation steps of the algorithms are explained above. Step 1 of both implementations is identical. In step 2, the procedures used for database reading are the same in both algorithm codes. Building the data structure for the algorithms is different from each other and additional step for modifying MFI is needed for Matrix Apriori. Candidate generation procedures for both algorithms are equal but counting support is clearly different.

4. Performance Evaluation

In this section, we compare Matrix Apriori and FP-Growth algorithms based on the publications discussed in previous chapter. Both algorithms are coded using Lazarus IDE (02.96.2) which uses Pascal programming language. ARTool (1.1.2) dataset generator is used for our synthetic datasets. Two case studies analyzing the algorithms are carried out step by step using two synthetic datasets generated in order i) to see their performance on datasets having different characteristics, ii) to understand the causes of performance differences in different phases. In order to keep the system state similar for all test runs, we assured all back-ground jobs which consume system resources were inactive. It is also ensured that test runs give close results when repeated.

4.1 Simulation Environment

The test runs are performed on a computer with 2.4 GHz dual core processor and 3 GB memory. At each run, both programs give results about data mining process. These are

- time cost for first scan of database,
- number of frequent items found at first scan of database,
- time cost for second scan of database and building the data structure,
- time cost for finding frequent itemsets,
- number of frequent itemsets found after mining process,
- total time cost for whole data mining process.

Procedures for first database scan are same for both algorithms so time cost is identical. During our case studies we will call first phase as first scan of database and second scan performed for building the specific data structure. Second phase will be traversing the data structures created by the first phase in order to find frequent itemsets.

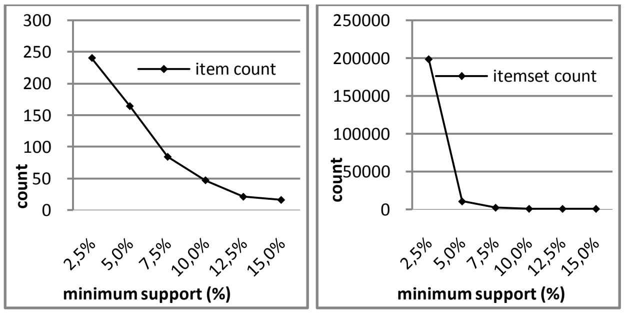
Although real life data has different characteristics from synthetically generated data as mentioned in [15], we used synthetic data since the control of parameters were easily manageable. In [16], the drawbacks of using real world data and synthetic data and comparison of some dataset generators are given. Our aim was to have datasets with different characteristics as representing different domain needs.

Synthetic databases are generated using ARtool software [17]. ARtool generates a database according to defined parameters as number of items, number of transactions, average size of transactions, number of patterns, average size of patterns. Two data sets are generated varying the parameters "number of items" and "average size of patterns" in order to have different dataset characteristics of different domains. One data set is characterized with long patterns and low diversity of items and other with short patterns and high diversity of items. These differences affect the size of the specific data structures of the algorithms and so the run times.

In the following subsections, performance analysis on the algorithms for two case studies is given. For the generated data sets, we aimed to observe how change of minimum support affects the performance of algorithms. The algorithms are compared for six minimum support values in the range of 15% and 2,5%.

4.2 Case1: Database of Long Patterns with Low Diversity of Items

A database is generated for having long patterns and low diversity of items where number of items=10000, number of transactions=30000, average size of transactions=20, average size of patterns=10. Number of frequent items is given in Figure 3a and number of frequent itemsets is given in Figure 3b while minimum support value is varied. It is clear that decrease in minimum support increases the number of frequent items from 16 to 240 and the number of frequent itemsets from 1014 to 198048.



(a) (b)

Figure 3. (a) Number of frequent items, (b) Number of frequent itemsets for Case1

The total performance of Matrix Apriori and FP-Growth is demonstrated in Figure 4. It is seen that their performance is identical for minimum support values above 7,5%. On the other hand below 7,5% minimum support value Matrix Apriori performs clearly better such that at 2,5% threshold it is 230% faster.

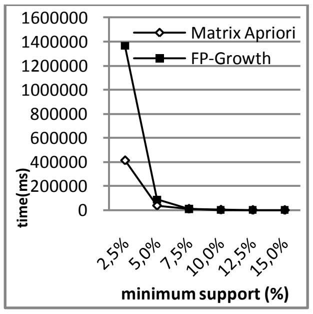
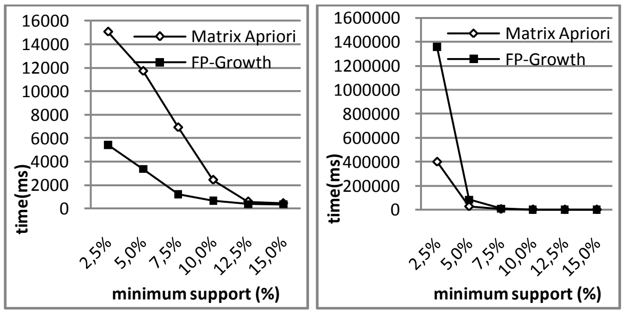


Figure 4. Total performance for Case1

The reason of FP-Growth's falling behind at total performance can be understood by looking at the performance of phases of evaluation. First phase performances of algorithms demonstrated in Figure 5a showed that building matrix data structure of Matrix Apriori needs 20% to 177% more time compared to building tree data structure of FP-Growth. First phase of Matrix Apriori shows similar pattern with the number of frequent items demonstrated in Figure 3a.



(a) (b)

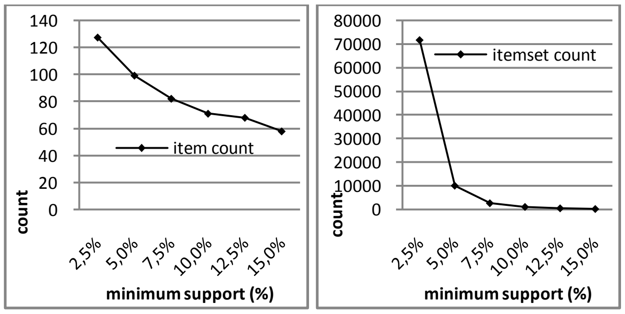
Figure 5. (a)First phase performance for Case1 (b) Second phase performance for Case1

The second phase of evaluation is finding frequent itemsets. As displayed in Figure 5b Matrix Apriori is faster at minimum support values below 10%. Although at 10% threshold, FP-Growth is 20% faster, Matrix Apriori is 240% faster at 2,5% threshold. As its expected, performance of second phases are related to number of frequent itemsets (see Figure 3b).

Our first case study showed that Marix Apriori performed better with decreasing threshold values for given database.

4.3 Case2: Database of Short Patterns with High Diversity of Items

A database is generated for short patterns and high diversity of items using the parameters where number of items=30000, number of transaction=30000, average size of transactions=20, average size of patterns=5. The change of frequent items and itemsets count is given in Figure 6a and Figure 6b consecutively. Frequent items found changes from 58 to 127 and frequent itemsets found changes from 254 to 71553 with decreasing minimum support values.



(a) (b)

Figure 6. a) Number of frequent items, (b) Number of frequent itemsets for Case2

The total performance of both algorithms is given in Figure 7. Increase in minimum support decreases runtime for both algorithms. For minimum support values 12,5% and 15% FP-Growth performed faster by up to 56%. However, for lower minimum support values Matrix Apriori performed better up to 150%.

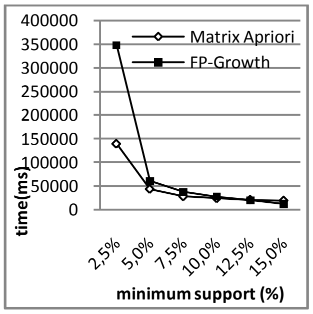
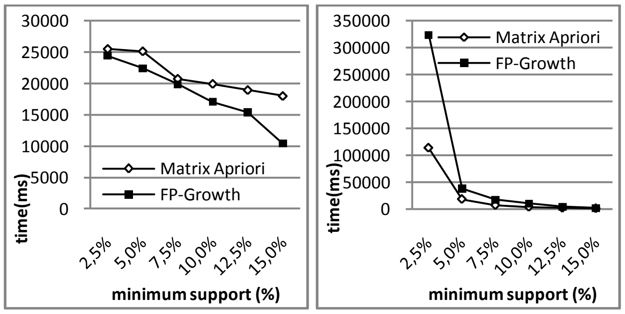


Figure 7. Total performance for Case2

First phase performance of algorithms is demonstrated in Figure 7a. FP-Growth is observed to have better first phase performance.



(a) **(b)**

Figure 7. (a)First phase performance for Case2 (b) Second phase performance for Case2

The second phase evaluation of algorithms as it is given in Figure 7b shows that Matrix Apriori performed better at all threshold values and the performance gap increases with decreasing threshold. This difference varies between 71% and 185%. Second phase performances of algorithms are related to number of frequent itemsets found like it was in first case study.

4.4 Discussion on Results

In this section, we analyzed the performance of FP-Growth and Matrix Apriori algorithms phase by phase, when minimum support threshold is changed. Two databases with different characteristics are used for our case studies. In both case studies, performances of algorithms are observed between minimum support values of 2,5% and 15%.

First case study is carried out on a database of long patterns with low diversity of items. It is seen that at 10%-15% minimum support values, performances of both algorithms are close. However, below 10% value, the performance gap between the algorithms becomes larger in favor of Matrix Apriori. Another point is that first phase of Matrix Apriori is affected from minimum support change more than FP-Growth. This is a result of increase in frequent items count. This increment affects building data structure step of Matrix Apriori dramatically. On the other hand, matrix data structure is faster leading to better total performance of Matrix Apriori.

Our second case study is performed on a database of short patterns with high diversity of items. It is seen that at 12,5%-15% minimum support values, performances of both algorithms are close. However, below 12,5% value, the performance gap between the algorithms becomes larger in favor of Matrix Apriori. It is seen that the impacts of having more items and less average pattern length caused both algorithms to have more runtime values compared to first case study. At 15% at first case study 1014 itemsets are found in 1031-1078 ms however at second case study 254 itemsets are found in 12172-19030 ms. In addition, for all threshold values first phase runtime values are higher in second case study.

Common points in both case studies are i) Matrix Apriori is faster at finding itemset phase compared to FP-Growth and slower at building data structure phase, ii) for threshold values below 10% Matrix Apriori is more efficient by up to 230%, iii) first phase performance of Matrix Apriori is correlated with number of frequent items, iv) second phase performance of FP-Growth is correlated with number of frequent itemsets.

5. Conclusion

In this paper, we benchmark and explain the FP-Growth and the Matrix Apriori association rule mining algorithms that work without candidate generation. Since the characteristics of data repositories of different domains vary, each algorithm is analyzed using two different synthetic databases consisting of different characteristics, i.e., one database has long patterns with a low diversity of items and the other database has short patterns with a high diversity of items.

Our case studies indicate that the performances of the algorithms are related to the characteristics of the given data set and the minimum support threshold applied. When the performances of the various algorithms are considered, we noticed that in constructing a matrix data structure, the Matrix Apriori takes more time in comparison to constructing the tree structure for the FP-Growth. On the other hand, during finding itemsets phase we discovered that the matrix data structure is considerably faster than the FP-Growth at finding frequent itemsets--thus retrieving and presenting the results in a more efficient manner.

We conclude, based on each of the case study that the Matrix Apriori algorithm performs better as the threshold decreases compared to FP-Growth, indicating that the former provides a better overall performance. To enhance our current findings, as our next step we plan to conduct a study that will help us to propose a new association rule mining algorithm that combines the strengths of both the Matrix Apriori and the FP-Growth algorithms.

References

[1] S. Kotsiantis, D. Kanellopoulos, Association Rules Mining: A Recent Overview, *International Transactions on Computer Science and Engineering, 32*(1), 2006, 71-82.

[2] M.H. Dunham, *Data Mining Introductory and Advanced Topics* (New Jersey Pearson Education, 2003).

[3] R. Agrawal, T. Imieliński & A. Swami, Mining Association Rules Between Sets of Items in Large Databases, *ACM SIGMOD Record*, *22*(2), 1993, 207-216

[4] R. Agrawal & R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases. *In Proceedings of the 20th international Conference on Very Large Data Bases*, San Francisco, CA, United States, 1994, 487-499

[5] M. Kantardzic, *Data Mining Concepts, Models, Methods, and Algorithms* (New Jersey, IEEE Press, 2003).

[6] A. Savasere, E. Omiecinski & S.B. Navathe, An Efficient Algorithm for Mining Association Rules in Large Databases. *In Proceedings of the 21th international Conference on Very Large Data Bases*, San Francisco, CA, United States, 1995, 432-444

[7] H. Toivonen, Sampling Large Databases for Association Rules. *In Proceedings of the 22th international Conference on Very Large Data Bases*, San Francisco, CA, United States, 1996, 134-145

[8] J. Han, J. Pei & Y. Yin, Mining Frequent Patterns without Candidate Generation, *ACM SIGMOD Record*, *29*(2), 2000, 1-12

[9] J. Pavón, S. Viana & S. Gómez, Matrix Apriori: Speeding up the Search for Frequent Patterns. *In Proceedings of the 24th IASTED international Conference on Database and Applications*, Innsbruck, Austria, 2006, 75-82

[10] N. Duru, An Application of Apriori Algorithm on a Diabetic Database. *In Proceedings of the 9th International Conference Knowledge-Based Intelligent Information and Engineering Systems*, Melbourne, Australia, 2005, 398-404

[11] R. Grossman, S. Kasif, R. Moore, D. Rocke & J. Ullman, Data Mining Research: Opportunities and Challenges. A Report of three NSF Workshops on Mining Large, Massive, and Distributed Data, available at http://www.rgrossman.com/dl/misc-001.pdf, 1998.

[12] X. Wu, V. Kumar & J. Ross Quinlan, Top 10 Algorithms in Data Mining, *Knowl. Inf. Syst. 14*(1), 2007, 1-37

[13] J. Han, M. Kamber, *Data Mining Concepts and Techniques* (San Diego, Academic Press, 2001).

[14] J. Hipp, U. Güntzer & G. Nakhaeizadeh, Algorithms for Association Rule Mining — A General Survey and Comparison. *SIGKDD Explor. Newsl. 2(1)*, 2000, 58-64.

[15] Z. Zheng, R. Kohavi & L. Mason, Real World Performance of Association Rule Algorithms. *In Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, United States, 2001, 401-406.

[16] A. Omari, R. Langer & S. Conrad, TARtool: A Temporal Dataset Generator for Market Basket Analysis. *In Proceedings of the 4th international Conference on Advanced Data Mining and Applications*, Chengdu, China, 2008, 400-410.

[17] L. Cristofor, *ARtool Project*. University of Massachusetts, Boston, available at http://www.cs.umb.edu/~laur/ARtool/