# Incremental Itemset Mining Based on Matrix Apriori Algorithm

Damla Oguz and Belgin Ergenc

Department of Computer Engineering, Izmir Institute of Technology, Izmir, Turkey
{damlaoguz,belginergenc}@iyte.edu.tr

**Abstract.** Databases are updated continuously with increments and re-running the frequent itemset mining algorithms with every update is inefficient. Studies addressing incremental update problem generally propose incremental itemset mining methods based on Apriori and FP-Growth algorithms. Besides inheriting the disadvantages of base algorithms, incremental itemset mining has challenges such as handling i) increments without re-running the algorithm, ii) support changes, iii) new items and iv) addition/deletions in increments. In this paper, we focus on the solution of incremental update problem by proposing the Incremental Matrix Apriori Algorithm. It scans only new transactions, allows the change of minimum support and handles new items in the increments. The base algorithm Matrix Apriori works without candidate generation, scans database only twice and brings additional advantages. Performance studies show that Incremental Matrix Apriori provides speed-up between 41% and 92% while increment size is varied between 5% and 100%.

**Keywords:** Itemset Mining, Matrix Apriori, Incremental Itemset Mining.

## 1 Introduction

Association rule mining, which was introduced by [1], has become a popular research area due to its applicability in various fields such as market analysis, forecasting and fraud detection. Given a market basket dataset, association rule mining discovers all association rules such as "A customer who buys item $X$, also buys item $Y$ at the same time". These rules are displayed in the form of $X \rightarrow Y$ where $X$ and $Y$ are sets of items that belong to a transactional database. Support of association rule $X \rightarrow Y$ is the percentage of transactions in the database that contain $X \cup Y$. Association rule mining aims to discover interesting relationships and patterns among items in a database. It has two steps; finding all frequent itemsets and generating association rules from the itemsets discovered. Itemset denotes a set of items and frequent itemset refers to an itemset whose support value is more than the threshold described as the minimum support.

Since the second step of the association rule mining is straightforward, the general performance of an algorithm for mining association rules is determined by the first step [2]. Therefore, association rule mining algorithms commonly concentrate on finding frequent itemsets. Apriori and FP-Growth are known to be the two important

A. Cuzzocrea and U. Dayal (Eds.): DaWaK 2012, LNCS 7448, pp. 192–204, 2012.

algorithms each having different approaches in finding frequent itemsets [3-4]. The Apriori Algorithm uses Apriori Property in order to improve the efficiency of the level-wise generation of frequent itemsets. On the other hand, candidate generation and multiple database scans are the drawbacks of the algorithm. FP-Growth comes with an approach that creates signatures of transactions on a tree structure to eliminate the need of database scans and outperforms compared to Apriori. A recent algorithm called Matrix Apriori which combines the advantages of Apriori and FP-Growth was proposed [5]. The algorithm eliminates the need of multiple database scans by creating signatures of itemsets in the form of a matrix. It provides a better overall performance than FP-Growth [6]. Although all these algorithms handle the problem of association rule mining, they ignore the dynamicity of the databases. When new transactions arrive, the entire process needs to be done from the beginning. The solution to this problem is incremental itemset mining, in which the idea is to keep frequent itemsets up-to-date with the arrival of increments to the database.

First group of incremental itemset mining algorithms are Apriori based [7-9]. The Fast UPdate Algorithm provides a solution to incremental association rule mining problem [7]. The frequencies of itemsets are determined by comparing itemsets in the original database and new transactions. The main goal of the algorithm is to reduce number of the candidate sets. In [8], a new incremental association rule mining algorithm was proposed. It uses a trie-like tree structure that stores the frequent 1-itemsets and 2-itemsets with their supports in the database. When new transactions arrive, frequent 1-itemsets and 2-itemsets are found and the trie is updated accordingly. Another algorithm in this group also avoids scanning the entire database when new transactions arrive, however, it does not handle minimum support change [9].

Second group of incremental itemset mining algorithms are FP-Growth based; they construct FP-tree incrementally. One of them was introduced by [10], which constructs a fast updated FP-tree (FUFP-tree) structure. It is similar to FP-tree, while the links between the nodes are bi-directional and it handles insertion and deletion of items. A new method, which constructs the FP-tree by assuming the minimum support as 1, was proposed by [11]. Therefore, when new transactions are added to the database, the tree is updated with scanning the increments twice.

Adaptation of the first and the second group of incremental itemset mining algorithms on stream data is also an intensive research area [12-13]. Analysis of the challenges brought by stream data and proposing a model for incremental itemset mining over stream data are beyond the scope of this paper.

Generally speaking, incremental frequent itemset mining has four challenges: i) adapting a base frequent itemset finding algorithm as to handle increments, ii) allowing new item appearances in increments, iii) being flexible to support changes during entire process and iv) handling deletions as well as additions in increments.

In this study, an incremental itemset mining algorithm, which is called Incremental Matrix Apriori, is proposed to provide solutions to these challenges. Since the base algorithm Matrix Apriori works without candidate generation and avoids multiple database scans, the proposed incremental algorithm inherits performance advantages. Performance evaluations show that the Incremental Matrix Apriori Algorithm is

significantly faster than re-running the Matrix Apriori Algorithm when new increments arrive.

This paper is organized as follows; Section 2 reviews incremental itemset mining algorithms. Section 3 presents the proposed algorithm for incremental itemset mining. In Section 4, test results and performance evaluations are discussed. Finally, this paper is concluded in Section 5 by raising several issues for future work.

## 2 Related Work

Association rule mining aims to discover frequent itemsets in a dataset. The Apriori Algorithm, which was proposed by [3], is one of the best-known association rule mining algorithms [14]. It uses prior knowledge of frequent itemset properties and runs an iterative approach called level-wise search. That is, $k$-itemsets are used to explore ($k$+1)-itemsets (they are called candidate itemsets before testing them against the database) by eliminating the candidates that do not satisfy the minimum support. The FP-Growth Algorithm handles the weaknesses of Apriori which are multiple scans of the database and candidate generation. It finds frequent itemsets without candidate generation by using a tree structure, called FP-tree, where each node stores an item with its number of occurrence in the database and a link to the next node [15].

The Matrix Apriori Algorithm offers a simple and efficient solution to the association rule mining. Database scan step is similar to FP-Growth whereas generating association rules is similar to Apriori. As a result, Matrix Apriori combines the two algorithms by using their positive properties [5]. FP-Growth and Matrix Apriori algorithms are compared with using different characteristics of data in [6].

All of the algorithms above ignore the dynamicity of the databases. However, transactional databases are dynamic in general. When new transactions arrive, these algorithms should be re-run in order to find up-to-date frequent itemsets. The solution to this problem is incremental frequent itemset mining. The Fast UPdate (FUP) Algorithm [7] is the first algorithm proposed for incremental mining of frequent itemsets. It handles the databases with transaction insertion only, relies on Apriori and uses the pruning techniques used in the Direct Hashing and Pruning Algorithm [16]. The main working principle of this algorithm can be summarized in two steps; scanning only new transactions to generate 1-itemsets, then comparing these itemsets with the previous ones and finally discovering all frequent itemsets of the same size iteratively. FUP2, an extended version of FUP, copes with both insertion and deletion of transactions, was proposed by [17].

A new algorithm, FOLDARM, which is suitable for incremental association rule mining, was presented by [8]. FOLDARM constructs a new data structure called Support-Ordered Trie Itemset, SOTrieIT (a trie-like tree structure). This structure only stores the frequent 1-itemsets and 2-itemsets with their supports in a descending order of support counts (the most frequent itemsets are found on the leftmost branches of the SOTrieIT) and is used to discover frequent 1-itemsets and 2-itemsets without scanning the database. When new transactions arrive, all frequent 1-itemsets and 2-itemsets are extracted from each transaction. The extracted information is used to

update the SOTrieIT without considering the support threshold. In order to mine frequent itemsets depth-first search is used starting from the leftmost first-level node. Unless a node that does not satisfy the support threshold, the traversal continues. Subsequently, the Apriori Algorithm is used to obtain other frequent itemsets.

The study by [9], proposed an incremental itemset mining algorithm based on Apriori which finds frequent itemsets and infrequent itemsets that are likely to be frequent after the arrival of new transactions. This algorithm uses the maximum support count of 1-itemsets in the database before the arrival of increments for finding potential frequent itemsets, called promising itemsets. In other words, in order to find a threshold value for finding promising itemsets, the maximum support count of 1-itemsets is used. It scans only new transactions, however it assumes that minimum support value does not change.

It is clear that when new transactions are added to the database, some frequent items may become frequent or infrequent. In the FP-Growth Algorithm, creating the FP-tree from the beginning according to the changes is time consuming. A fast upda7ted FP-tree (FUFP-tree) structure and an incremental FUPF-tree maintenance algorithm was proposed by [10]. However, when a sufficiently large number of transactions are inserted to the database, the whole FUFP-tree should be re-constructed [18].

Another method for constructing FP-tree incrementally was presented by [11]. The minimum support threshold is accepted as 1 and FP-tree is updated by scanning the new transactions twice. Five synthetic and one real datasets are used in the experiments with different number of items and transactions. In both cases, this approach performs better compared to building the tree from the beginning.

The comparison of incremental itemset mining algorithms is displayed in Table 1. All these algorithms can handle the maintenance problem in case of insertion and new items can be presented in the increments. FOLDARM, Incremental FP-tree and Incremental Matrix Apriori can handle minimum support change while FUP, FUP2, FUFP-tree and Promising Frequent Itemset cannot. Also FUP, FUP2 and Promising Frequent Itemset need candidate generation. There is a point that should be taken into consideration when comparing these algorithms; FOLDARM only addresses finding 1-frequent itemsets and 2-frequent itemsets.

**Table 1.** Comparison of Incremental Itemset Mining Algorithms

| | Deletion | Support Change | New Item Occurrence | No Candidate Generation |
|---|---|---|---|---|
| FUP [7] | | | + | |
| FUP2 [17] | + | | + | |
| FOLDARM [8] | + | + | + | + |
| Promising Frequent Itemset [9] | | | + | |
| FUFP-tree [10] | + | | + | + |
| Incremental FP-tree [11] | + | + | + | + |
| Incremental Matrix Apriori | | + | + | + |

# 3 Incremental Itemset Mining Algorithm

The Incremental Matrix Apriori Algorithm is proposed to overcome the problem of mining frequent itemsets in dynamically updated databases. As seen in Table 1, besides finding frequent itemsets by scanning only new transactions, this algorithm handles new items in the updates and allows support threshold changes. This chapter is divided into two subsections. The first one explains the base algorithm Matrix Apriori and the second one presents the Incremental Matrix Apriori Algorithm.

## 3.1 Matrix Apriori Algorithm

In Matrix Apriori, the frequent items are stored in a matrix called MFI (Matrix of Frequent Items) and the supports of the itemsets are stored in a vector called STE. Initially the database is scanned in order to determine the frequent items and the frequent items list. Subsequently, in the second scan, MFI and STE are built as follows. Beginning from the second row of the matrix, the matrix is stored according to the transactions. If the transaction contains the item, the value "1", if not, the value "0" is stored in the MFI and the STE value is set to "1". If the transaction is already included in the MFI, then it is not stored in a new row, but its STE is incremented by 1.

After the construction of the matrix, it is modified in order to speed up the frequent itemset search. For each column, beginning from the first line, each cell value is set to the number of the line where the cell value is equal to "1" in the unmodified matrix. If there are not any values of "1" in the remaining lines of the unmodified matrix, the cell value is set to "1". After the matrix construction, frequent itemsets are found as follows. Beginning from the item that has the least support value; the item is compared with the items found on its left in order to find frequent itemsets. Following that, their support values are counted. The support value of an itemset is found by sequentially adding the related rows of STE from top to bottom.

## 3.2 Incremental Matrix Apriori Algorithm

In order to provide incremental mining of frequent itemsets, the matrix is constructed by the minimum support value of 1. That means all items are kept in the MFI without considering their frequencies. Doing so, flexibility for support change is enabled as well. Due to the structure of the matrix, items are kept in a descending support order in the MFI. So finding frequent itemsets with any support threshold is easy. Since all items are kept in the MFI, frequent itemsets can be calculated from the item that satisfies the minimum support.

The process before the arrival of increments can be seen in Figure 1. The database is scanned and the support counts of items are calculated. The list of items in the specified order is named as 1-itemset ordered lists of items. All items in the transactions are included to the 1-itemset ordered list of items without considering if the item's support count is more than the minimum support or not. This process is shown in Figure 1.a. Afterwards, the MFI is constructed and then modified as in Figure 1.b. The way of construction of the MFI is the same as Matrix Apriori.

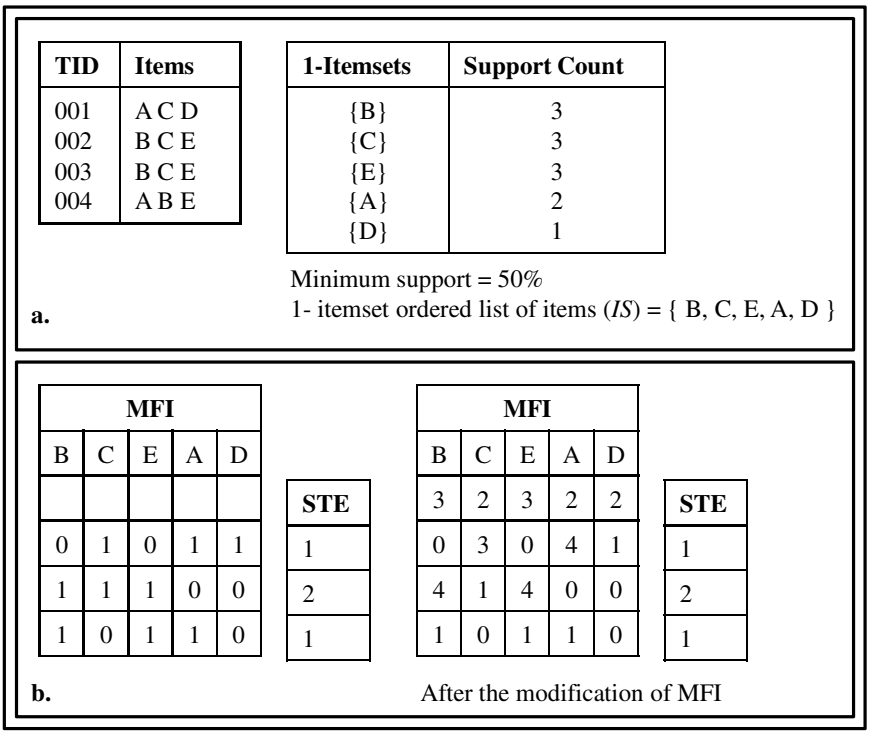
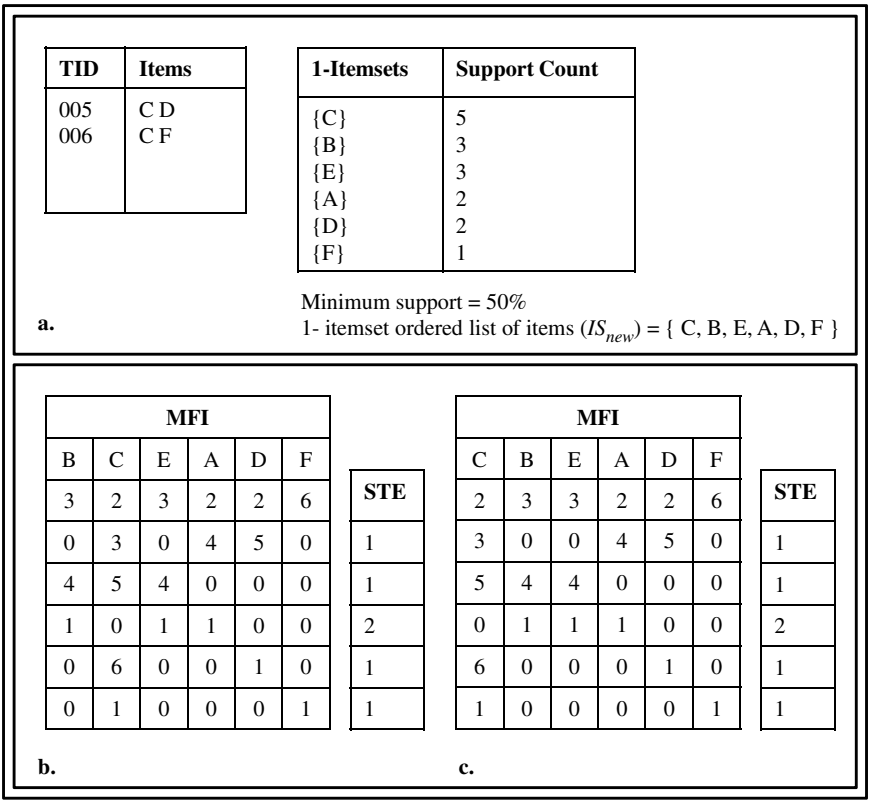



**Fig. 1.** Incremental Matrix Apriori – Before Increments

When new transactions arrive, they are scanned and 1-itemset ordered list of items are updated as in Figure 2.a and as indicated in lines 1-2 of pseudo code in Figure 3. The new items in increments are included to the MFI by adding new columns as in lines 3-5 in Figure 3. The MFI is updated as follows. First, the new transaction is checked whether it exists in the MFI or does not. If it exists, its STE is incremented by 1; if it does not exist, it is added to the MFI. Adding to the MFI is done by setting the cell value to the line number of transaction where value "1" is stored in the MFI. When the item does not exist in the remaining transactions of the incremental database, the cell value is set to "1". This entire process is shown in Figure 2.b and in lines 6-12 in Figure 3. Finally, according to the change of the 1-itemset ordered list of items, the order of items in the MFI is changed as in Figure 2.c and in line of 13 in Figure 3.

Since Matrix Aprori does not have an update feature, it runs from the beginning on the updated database whereas Incremental Apriori only runs on the updates when new transactions arrive. Needless to say, Incremental Matrix Apriori finds exactly the same frequent itemsets as the Matrix Apriori does. Besides avoiding database scan and avoiding the construction of different matrices for mining frequent items with different support thresholds, management of matrix is easy. Moreover, the base algorithm scans database only twice and does not generate candidate sets.



| TID | Items |
|---|---|
| 005 | C D |
| 006 | C F |

| 1-Itemsets | Support Count |
|---|---|
| {C} | 5 |
| {B} | 3 |
| {E} | 3 |
| {A} | 2 |
| {D} | 2 |
| {F} | 1 |

Minimum support = 50%

1- itemset ordered list of items ($IS_{new}$) = { C, B, E, A, D, F }

**a.**

| MFI | | | | | | |
|---|---|---|---|---|---|---|
| B | C | E | A | D | F | STE |
| 3 | 2 | 3 | 2 | 2 | 6 | 1 |
| 0 | 3 | 0 | 4 | 5 | 0 | 1 |
| 4 | 5 | 4 | 0 | 0 | 0 | 2 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 6 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**b.**

| MFI | | | | | | |
|---|---|---|---|---|---|---|
| C | B | E | A | D | F | STE |
| 2 | 3 | 3 | 2 | 2 | 6 | 1 |
| 3 | 0 | 0 | 4 | 5 | 0 | 1 |
| 5 | 4 | 4 | 0 | 0 | 0 | 2 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 6 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |

**c.**

**Fig. 2.** Incremental Matrix Apriori – After Increments

```
INPUT: MFI, STE, Additions A, 1-itemset ordered list IS
OUTPUT: MFI, STE, 1-itemset ordered list IS_new
BEGIN
1  Scan A
2  Create IS_new from IS using A
3  FOR each new item in IS_new
4    Add a column to MFI
5  END
6  FOR each transaction in A
7    IF the transaction exists in MFI
8       Update the STE by incrementing by 1
9    ELSE IF the transaction does not exist in MFI
10      Add a new line to MFI
11      Set the STE of the new line to 1
12 END
13 Update and reorder MFI using IS_new
14 Return IS_new, MFI, STE
END
```

**Fig. 3.** Update Process in Incremental Matrix Apriori Algorithm

# 4 Performance Evaluation

In this section, the Incremental Matrix Apriori Algorithm is compared with the Matrix Apriori Algorithm when new transactions are added to the database. Both algorithms are implemented in Java and test runs are performed on a computer with 2.93 GHz dual core processor and 2 GB memory. Two synthetic datasets with different characteristics are used; they are generated by utilizing ARTool (1.1.2) dataset generator [6]. Difficulties in finding real datasets compelled the tests to be done with synthetic datasets.

In the following subsections, performance analysis of the algorithms for two case studies while varying the size of incremental dataset is given. The purpose is to observe how the percentage of the increment sizes affects the performance of the algorithms for the generated datasets. The algorithms are compared for 20 increasing addition sizes in the range of 5% and 100%. In these tests, the minimum support is 10% and the initial database has 15000 transactions for both case studies. During performance evaluations, it is ensured that the system state is similar in all test runs and they give close results when repeated.

## 4.1 Case1: Database of Long Patterns with Low Diversity of Items

The first dataset has the following characteristics i) long patterns with low diversity of items, ii) number of items is 10000, iii) average size of transactions is 20 and iv) average size of patterns is 10. The performances of Matrix Apriori and Incremental Matrix Apriori with different increment sizes are demonstrated in Figure 4. In every increment size, Incremental Matrix Apriori performs better than re-running Matrix Apriori.

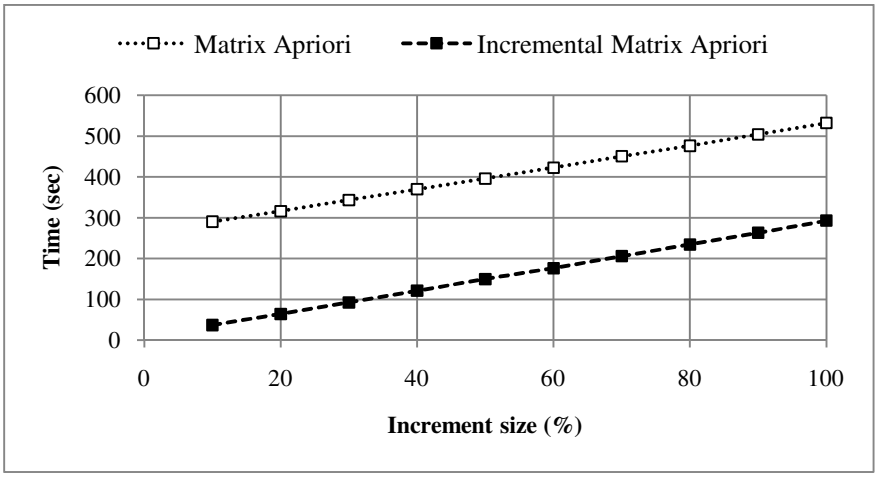



**Fig. 4.** Run-time with Different Increment Sizes for Case 1

In Figure 5, the speed-up with different increment sizes is shown. The speed-up increases from 45% to 92% as the increment size decreases. Although the speed-up decreases as the increment size increases, Incremental Matrix Apriori is 45% faster than re-running Matrix Apriori.

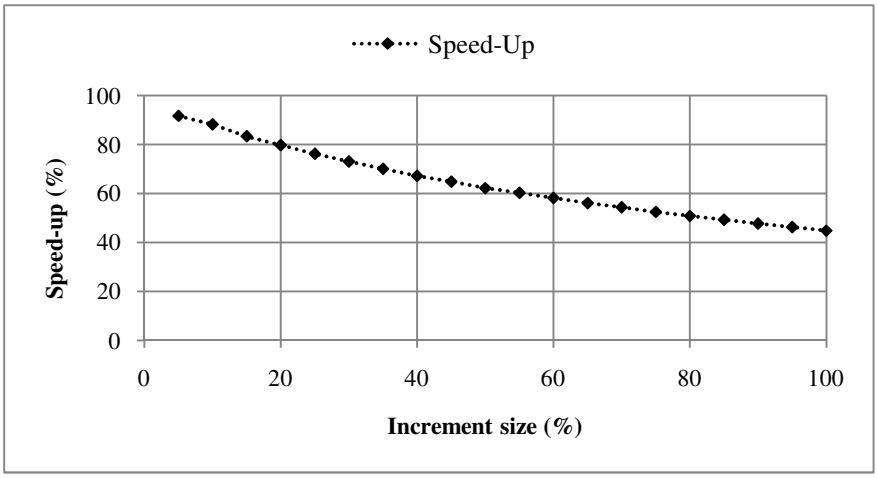



**Fig. 5.** Speed-up with Different Increment Sizes for Case 1

### 4.2 Case2: Database of Short Patterns with High Diversity of Items

The second dataset is composed of the following characteristics i) short patterns and high diversity of items, ii) number of items is 30000, iii) average size of transactions is 20 and v) average size of patterns is 5. The performances of Matrix Apriori and Incremental Matrix Apriori with different increment sizes are demonstrated in Figure 6.

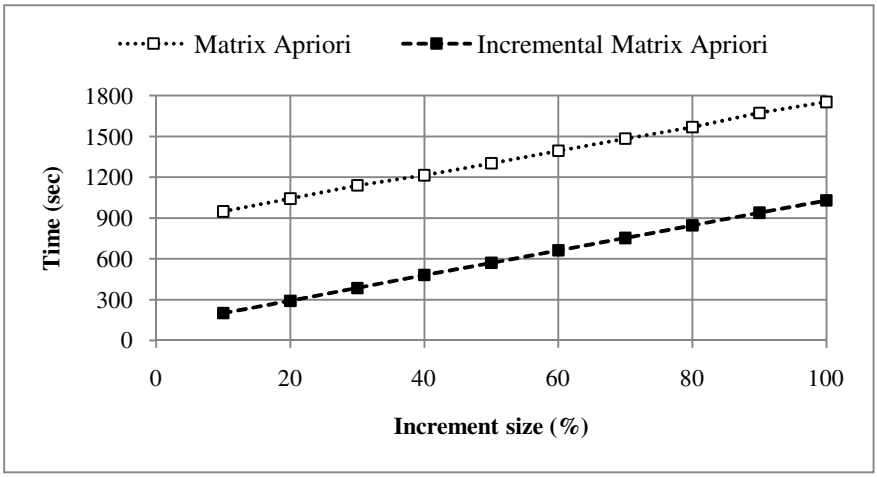



**Fig. 6.** Run-time with Different Increment Sizes for Case 2

As expected, in every increment size, Incremental Matrix Apriori outperforms the Matrix Apriori Algorithm as in Case 1.

The speed-up in different increment sizes is illustrated in Figure 7. The decrease in increment size increases the speed-up percentage from 41 to 81. Although there is a decrease in speed-up when the increment size becomes larger, Incremental Matrix Apriori is almost 41% faster than re-running Matrix Apriori.

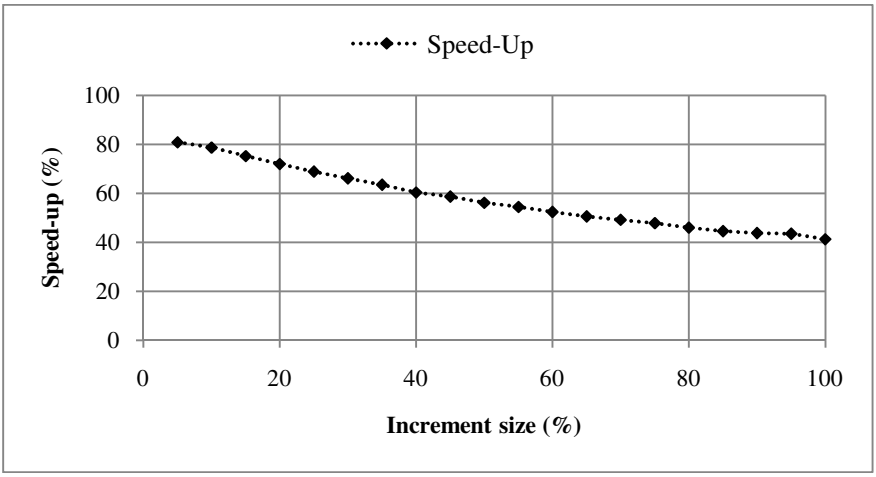



**Fig. 7.** Speed-up with Different Increment Sizes for Case 2

Both the test results in Case 1 and Case 2 reveal that Incremental Matrix Apriori performs better than running Matrix Apriori with each update in the range of 5% and 100%.

### 4.3 Discussion on Results

The test results demonstrate that, when the new transactions are added to database, Incremental Matrix Apriori finds frequent itemsets more efficiently than re-running Matrix Apriori. Two databases with different characteristics are used and the minimum support is given as 10% in both cases.

Performances of Incremental Matrix Apriori by using the first dataset and second dataset are shown in Figure 8. Finding frequent itemsets for Case 2 takes longer than Case 1. There are 10000 items in Case 1 while there are 30000 items in Case 2. This result has been expected because Incremental Matrix Apriori keeps all items in the MFI. The run-time increases as does the number of items. Moreover, the relationship between increment size and time are linear both two cases.

Figure 9 shows the speed-up comparison of Incremental Matrix Apriori for Case 1 and Case 2. Although the behavior in two cases is similar, the speed-up for Case 1 is higher than that of Case 2. This may be due to the number of items. Since Incremental Matrix Apriori keeps all the items in the matrix even it exists in only in one transaction, when the number of items increases, the total time increases as well. On the other hand, Matrix Apriori keeps the items that satisfy the minimum support. So the speed-up is lower in Case 2.

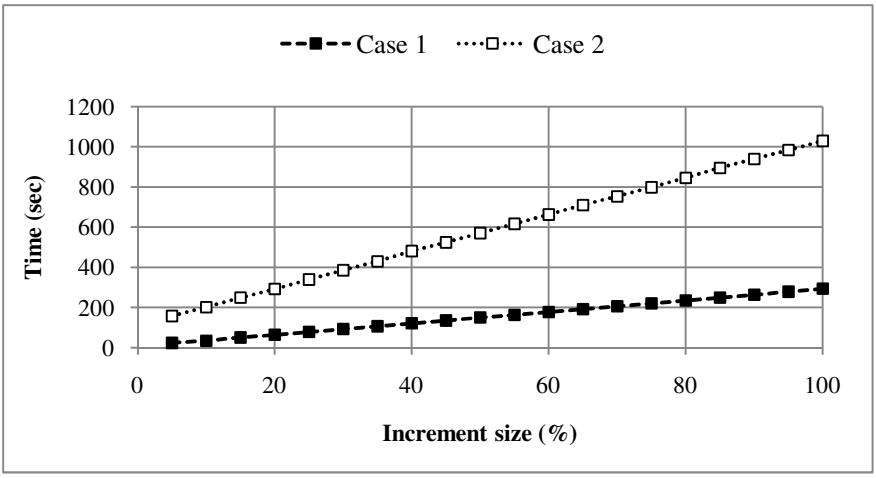



**Fig. 8.** Run-time for Case 1 and Case 2

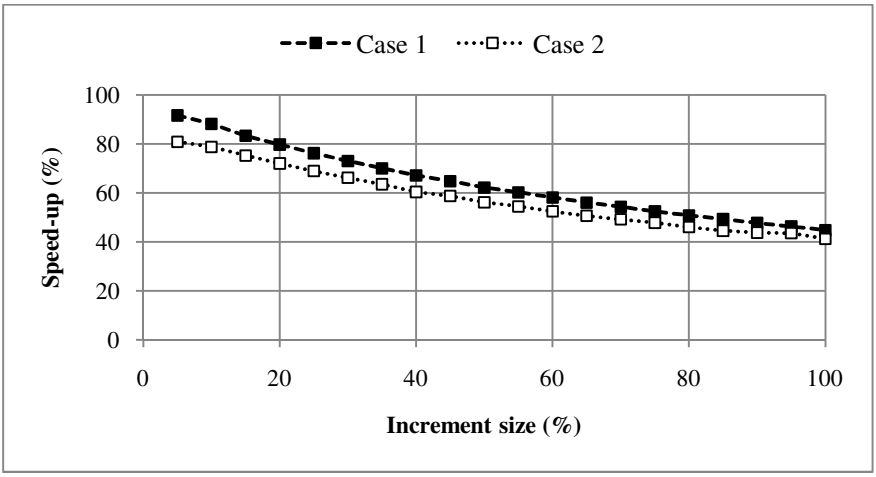



**Fig. 9.** Speed-up for Case 1 and Case 2

## 5 Conclusion and Future Work

In this paper, we propose a new algorithm for the problem of incremental itemset mining which is based on the Matrix Apriori Algorithm. A matrix structure is used in order to handle new transactions. Accepting the minimum support as 1, all items are kept in the matrix in a descending order of support counts. Therefore, without scanning the entire database, the new transactions and new items can be added to the matrix easily. Moreover, this approach allows the user to change minimum support. Since all frequent and infrequent items exist in the matrix, the calculation of frequent itemsets can start from the updated minimum support count.

Experimental results show that Incremental Matrix Apriori performs better than re-running Matrix Apriori for handling new transactions. The algorithms are compared for 20 increment sizes in the range of 5% and 100% by using two datasets with different characteristics. When the number of items or the increment size decreases, the speed-up by Incremental Matrix Apriori increases. However, both tests show that Incremental Matrix Apriori provides speed-up between 41% and 92% while increment size is varied between 5% and 100%.

Our work is ongoing and we aim to extend our algorithm to handle the deletion of transactions as well. After this step, our plan is to compare our incremental approach with other incremental itemset mining approaches in order to understand its strengths and weaknesses. In addition, we are planning to test our algorithm on real datasets.

## References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. ACM SIGMOD Record 22(2), 207–216 (1993)
2. Han, J., Kamber, M.: Data mining: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems. Elsevier, San Francisco (2006)
3. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: 20th International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994)
4. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. ACM SIGMOD Rec. 29(2), 1–12 (2000)
5. Pavon, J., Viana, S., Gomez, S.: Matrix Apriori: Speeding Up the Search for Frequent Patterns. In: 24th IASTED International Conference on Database and Applications, pp. 75–82. ACTA Press, Anaheim (2006)
6. Yıldız, B., Ergenç, B.: Comparison of Two Association Rule Mining Algorithms without Candidate Generation. In: 10th IASTED International Conference on Artificial Intelligence and Applications, Innsbruck (2010)
7. Cheung, D., Han, J., Ng, V., Wong, C.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In: 12th International Conference on Data Engineering, pp. 106–114. IEEE Computer Society, Washington (1996)
8. Woon, Y.K., Ng, W.K., Das, A.: Fast online dynamic association rule mining. In: 2nd International Conference, vol. 1, pp. 278–287 (2001)
9. Amornchewin, R., Kreesuradej, W.: Incremental Association Rule Mining Using Promising Frequent Itemset Algorithm. In: 6th International Conference on Information, Communications & Signal Processing, pp. 1–5 (2007)
10. Hong, T.P., Lin, C.W., Wu, Y.L.: Incrementally Fast Updated Frequent Pattern Trees. Expert Syst. Appl. 34(4), 2424–2435 (2008)
11. Muhaimenul, A.R., Barker, K.: Alternative Method for Incrementally Constructing the FP-tree. In: Chountas, P., Petrounias, I., Kacprzyk, J. (eds.) Intelligent Techniques and Tools for Novel System Architectures. SCI, vol. 109, pp. 361–377. Springer, Heidelberg (2008)
12. Jin, R., Agrawal, G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. In: 5th IEEE International Conference on Data Mining, pp. 210–217. IEEE Computer Society, Washington (2005)
13. Calders, T., Dexters, N., Goethals, B.: Mining Frequent Itemsets in a Stream. In: 7th IEEE Conference on Data Mining, pp. 83–92. IEEE Computer Society, Washington (2007)

14. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 Algorithms in Data Mining. Knowl. Inf. Syst. 14(1), 1–37 (2007)
15. Kotsiantis, S., Kanellopoulos, D.: Association Rules Mining: A Recent Overview Basic Concepts & Basic Association Rules Algorithms. Science 32(1), 71–82 (2006)
16. Park, J.S., Chen, M.S., Yu, P.S.: An Effective Hash-Based Algorithm for Mining Association Rules. SIGMOD Rec. 24(2), 175–186 (1995)
17. Cheung, D.W.L., Lee, S.D., Kao, B.: A General Incremental Technique for Maintaining Discovered Association Rules. In: 5th International Conference on Database Systems for Advanced Applications, pp. 185–194. World Scientific Press (1997)
18. Lin, C.-W., Hong, T.-P., Lu, W.-H., Chien, B.-C.: Incremental Mining with Prelarge Trees. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 169–178. Springer, Heidelberg (2008)