DEVELOPMENT OF AN APPLICATION FOR DYNAMIC ITEMSET MINING UNDER MULTIPLE SUPPORT THRESHOLDS

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
Izmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by

Nourhan ABUZAYED

July 2016

İZMİR

We approve the thesis of **Nourhan Nagee ABUZAYED**

Examining Committee Members:

_____

Asst. Prof. Dr. Belgin Ergenç BOSTANOĞLU
Department of Computer Engineering
İzmir Institute of Technology

_____

Asst. Prof. Dr. Selma TEKİR
Department of Computer Engineering
İzmir Institute of Technology

_____

Assoc. Prof. Dr. Adil ALPKOÇAK
Department of Computer Engineering
Dokuz Eylül University

July 2016

_____

Asst. Prof. Dr. Belgin Ergenç BOSTANOĞLU

Supervisor, Department of Computer Engineering
İzmir Institute of Technology

_____

Assoc. Prof. Dr. Y. Murat ERTEN
Head of the Department of
Computer Engineering

_____

Prof. Dr. Bilge KARAÇALI
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

I would like to express my sincere thanks and appreciation to my supervisor Asst.Prof.Dr.Belgin Ergenç Bostanoğlu for her guidance, patience, unique support and contributions during my master period.

This work is partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under ARDEB 3501 Project No: 114E779. I would like to express my sincere gratitude for the support provided.

I would like to thank my colleages in the project Sadeq Darrab and Cumhur Ozturk for their help.

Finally, I would like to give special thanks to my husband, without his encouragement, I would not have finished this thesis.

ABSTRACT

DEVELOPMENT OF AN APPLICATION FOR DYNAMIC ITEMSET MINING UNDER MULTIPLE SUPPORT THRESHOLDS

Handling dynamic aspect of databases and multiple support threshold requirement of items are two important challenges of frequent itemset mining algorithms. Frequent itemsets should be updated when the database is updated without re-running the mining algorithm. Frequent itemset mining algorithm should consider different support thresholds in order not to cause rare item problem. Existing dynamic itemset mining algorithms are devised for single support threshold whereas multiple support threshold algorithms are static. This thesis focuses on dynamic update problem of frequent itemsets under multiple support thresholds and introduces *Dynamic MIS1* and *Dynamic MIS2* algorithms. They are i) tree based and scan the database once, ii) consider multiple support thresholds, and iii) handle increments of additions, additions with new items and deletions. Proposed algorithms are compared to CFP-Growth++ and findings are; in static databases 1) *Dynamic MIS1* achieves up to 5 times speed-up against CFP-Growth++ since it does not require tree pruning and merging, 2) execution time of *Dynamic MIS2* and CFP-Growth++ are similar, 3) memory usage of *Dynamic MIS1* is higher than CFP-Growth++, since it keeps whole tree in memory, in dynamic database 1) *Dynamic MIS1* and *Dynamic MIS2* perform better than CFP-Growth++ since they run only on increments, 2) *Dynamic MIS1* can achieve speed-up of 56 times against CFP-Growth++, whereas the speed-up of *Dynamic MIS2* cannot exceed 2 times, 3) *Dynamic MIS2* is slightly better than CFP-Growth++ until increment size is less than 85% when the database is large and sparse, 25% when the database is small and dense.

ÖZET

ÇOKLU DESTEK EŞIKLERINDE DINAMIK SIK KÜMELER MADENCILIĞI IÇIN UYGULAMA GELIŞTIRILMESI

Veritabanlarının devingenliği ve kümelerin farklı destek eşiklerine olan gereksinimi, sık kümeler madenciliği algoritmalarının önemli iki zorluğudur. Veritabanına gelen her güncellemede, sık kümelerin tüm algoritmanın baştan çalıştırılmasına gerek kalmadan güncellenebilmesi ve seyrek kümeler problemine yol açmayacak şekilde kümelerin farklı eşik değerlerine olan gereksiniminin dikkate alınması gerekmektedir. Mevcut algoritmalar ya güncellemeleri sık kümelere yansıtmaya ya da farklı eşik değerlerini dikkate almaya odaklanmışlardır. Bu tez; veritabanlarının devingen güncellenmeleri durumunda, sık kümelerin de güncellenmesine ve sık kümelerin farklı eşik değerleri gözedilerek bulunmasına yoğunlaşmış ve *Dynamic MIS1* and *Dynamic MIS2* algoritmalarını önermiştir. Bu algoritmalar i) ağaç tabanlıdır ve veritabanını sadece bir kere tarar, ii) çoklu eşik değerlerini dikkate alır ve iii) eklemeli, yeni elemanla eklemeli ve silmeli güncellemelerde sık kümeleri güncelleyebilirler. Önerilen algoritmalar CFP-Growth++ algoritması ile karşılaştırılmış ve şunlar bulunmuştur; statik veritabanlarında 1) *Dynamic MIS1*, CFP-Growth++'dan 5 kata kadar daha hızlıdır çünkü ağaç budama ve birleştirme yapmamaktır, 2) *Dynamic MIS2 ve* CFP-Growth++ algoritmalarının çalışma zamanları yakındır, 3) *Dynamic MIS1'ın* bellek gereksinimi tüm ağacı tutması gerektiği için CFP-Growth++'dan daha fazladır, devingen veritabanlarında ise 1) sadece gelen güncelleme üzerinde çalıştıkları için *Dynamic MIS1* and *Dynamic MIS2* algoritmaları CFP-Growth++'dan hızlıdırlar, 2) *Dynamic MIS1'ın* hızlanması 56 kata kadar ulaşırken, *Dynamic MIS2'inki* 2 katı geçemez, 3) geniş ve seyrek veritabanında gelen güncellemenin büyüklüğü % 85'i, küçük ve sık veritabanlarında ise % 25'i geçmediği durumlarda *Dynamic MIS2* algoritması CFP-Growth++'dan daha etkindir.

DEDICATION

I would like to dedicate this work to my mother’s soul, my father, my husband and my sweet children.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

Lately intensive research focused on association rule mining, which is one of the main tasks of data mining. Association Rule Mining has received greater focus because of its applicability in decision support, share market, layout of shelves in supermarkets, web log analysis, text mining, understanding customer behavior, telecommunication alarm diagnosis, and prediction [19].

Association rule was first introduced by Agarwal et al. [12] and is defined as, X% of the customers who buy item A also buy item B (formulated as A→ B). Therefore association rules are meant to find the impact of one set of items on another set of items. Association rule mining has two problems: (1) finding frequent itemsets/patterns, (2) generating association rules [13]. But the first problem is the more challenging so it is a focus of many studies. Many powerful algorithms have been proposed to find the frequent itemsets from massive databases. The most classical one is the Apriori algorithm [13] that uses candidate generation and testing approach to discover frequent itemsets. Later on other algorithms using Apriori-like technique were introduced in [20, 21, 22, 23, 24, 25, 26, 27, and 28]. Because of the drawback of candidate generation and multiple hits on the database in Apriori-like algorithms, algorithms that do not depend on candidate generation were introduced, such as FP-Growth [14] and Matrix Apriori [15].

The major drawback of the above algorithms is their dependence on single user given support value. Single support is not enough because of the rare item problem [18] since all items in the data does not have similar frequencies. In many applications, some items appear very frequently in the data, while others rarely appear. If minsup is set too high, those rules that involve rare items will not be found. To find rules that involve both frequent and rare items, minsup has to be set very low. This may cause combinatorial explosion because those frequent items will be associated with one another in all possible ways [17]. As a result of this. Some algorithms like MSapriori[17], CFP-Growth[2], CFP-Growth++[3] and MISFP-Growth [36] are introduced to solve the problem of itemset mining under multiple support threshold.

One main assumption in all the above-mentioned algorithms is that database is static, but in real life, databases are constantly updated with new data, and old data may as well be deleted or modified. This implies, the originally discovered association rule may no longer be valid and yet new interesting rules may emerge as a result of an update on the database. The most straightforward way to update the rules would be to repeat the entire mining process from scratch however this is very expensive in terms of execution time and memory allocation. Therefore many efficient algorithms both dependent on candidate generation and non-candidate generation techniques have been introduced in [6, 7, 29, 30, 31 and 32]. These algorithms perform faster and use less system resources than repeating the processes from scratch.

The previous works handle either the dynamic itemset mining with single support threshold or static itemset mining with multiple support thresholds. In this thesis, we introduce two new algorithms; *Dynamic MIS1* and *Dynamic MIS2* algorithms. They provide a solution to the dynamic itemset mining under multiple support thresholds problem. The two algorithms are tree based structure, scan the database only once and avoid the candidate generation problem, also they handle increments with additions, increments with additions with new items and increments with deletions.

The organization of this thesis is as follows:

Chapter 2 is the related work that gives general information about association rule mining, frequent itemset mining. Existing important algorithms for frequent itemset mining under multiple support thresholds are presented, followed by a review of dynamic itemset mining algorithms and finally an algorithm for dynamic itemset under multiple support threshold is mentioned. Chapter 3 proposes Dynamic MIS1 and Dynamic MIS2 Algorithms. First, the base algorithm CFP-Growth++ is explained, and then the Dynamic MIS1and Dynamic MIS2 Algorithms are introduced with several examples. These examples show how each proposed algorithm handles the tree building, additions, additions with new items and finally they demonstrate how the deletion is handled. Chapter 4 shows the performance evaluation. The chapter begins with a presentation of the dataset properties followed by the complexity analysis of the algorithms. Then, the performance of our static and dynamic algorithms are compared with the base algorithm CFP-Growth++, followed by the last sub section for the

discussion on results. Chapter 5 is the conclusion chapter. A summary of this thesis is stated and the future work.

CHAPTER 2

RELATED WORK

Association rule mining aims to discover the relationships and the patterns in a dataset by including two steps: i) finding all frequent itemsets and ii) generating association rules from those frequent itemsets. The frequency of an itemset is also referred to as the support count, which is the number of transactions that contain the itemset. An itemset is named as frequent itemset if its support count satisfies the minimum support threshold [34]. Minimum support and minimum support threshold are used interchangeably. Confidence, which assesses the strength of an association rule, is another measure for defining association rules. The confidence for an association rule X $\rightarrow$ Y is the ratio of transactions that contain X $\cup$ Y to the number of transactions that contain X [12, 14]. A formal definition of association rule mining is:

Given a set of items $I = \{I_1, I_2, \ldots, I_m\}$ and a database of transactions $D = \{T1, T2, \ldots, T_n\}$ where each transaction T is a set of items such that T $\subseteq$ I, and X, Y are set of items, the association rule mining problem is to identify all association rules $X \rightarrow Y$ with a minimum support and confidence, where support of association rule $X \rightarrow Y$ is the percentage of transactions in the database that contain $X \cup Y$, and confidence is the ratio of support of $X \cup Y$ to support of X [14].

2.1 Association Rule Mining Algorithms

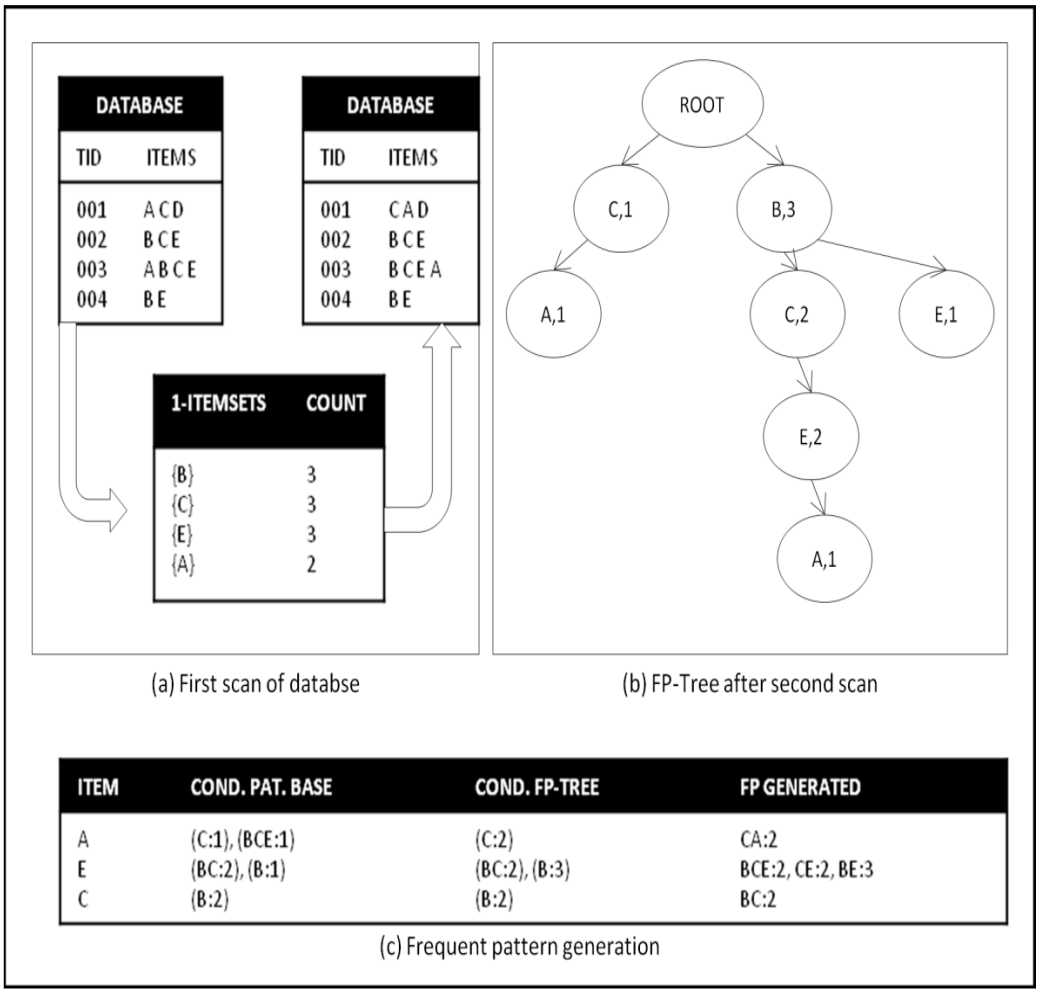
The Apriori Algorithm is one of the best-known association rule mining algorithms [12]. It uses prior knowledge of frequent itemset properties and runs an iterative approach called level-wise search. That is, k-itemsets are used to explore (k +1)-itemsets (they are called candidate itemsets before testing them against the database) by eliminating the candidates that do not satisfy the minimum support. This process terminates when no frequent or candidate set can be generated. The efficiency of the level-wise generation of frequent itemsets is improved by the Apriori Property: "All nonempty subsets of a frequent itemset must be frequent". By means of this

property, many unnecessary candidate generation and support counting are eliminated [34]. This property is used in many other association rule mining algorithms such as Fast Update Algorithm [6], Fast Update 2 Algorithm [7], FP-Growth Algorithm [14] and Matrix Apriori Algorithm [15].

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. FP-tree creation is shown in Figure 2.1. First, frequent items are determined from the database as in Figure 2.1.a and then the tree is constructed as in Figure 2.1.b. A header table, in which frequent items with their support counts are kept in a descending order of support counts, is built to simplify tree traversal. The frequent itemsets are discovered with only two scans over the database. The first scan is for getting frequent 1-itemsets and their support counts same as the Apriori Algorithm and the second one is for generating the FP-tree. When the minimum support decreases, the length of frequent items and the number of candidate items increase consequently in Apriori. Therefore, FP-Growth performs better than Apriori when minimum support value is decreased [14].

Here is an example for the FP-Growth as presented in [16]. 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.

In Figure 2.1 FP-Growth algorithm is visualized for an example database with minimum support value 2 (50%).



(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 2.1. FP-Growth example.

First, a scan of database derives a list of frequent items in descending order (see Figure 2.1a). 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 2.1b, we can see the transactions and the tree constructed [16].

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 from discovered patterns is similar to Apriori. As a result, Matrix Apriori combines the two algorithms by using their positive properties [15]. [16] Compared FP-Growth and Matrix Apriori algorithms by using different characteristics

of data and found that the total performance of the Matrix Apriori is better than FP-Growth for minimum support values below 10%.

2.2 Association Rule Mining with Multiple Support Thresholds Algorithms

The major drawback of the above algorithms is their dependence on single minimum support value assuming that all the items in the data have the same nature or similar frequencies. So, Single support is not enough because of the rare item problem [18] since all items in the data does not have similar frequencies. In many applications, some items appear very frequently in the data, while others rarely appear. If minsup is set too high, those rules that involve rare items will not be found. To find rules that involve both frequent and rare items, minsup has to be set very low. This may cause combinatorial explosion because those frequent items will be associated with one another in all possible ways [17]. Some algorithms like MSapriori [17] , CFP-Growth [2], CFP-Growth++ [3] and MISFP-Growth [36], are introduced to solve the problem of itemset mining under multiple support thresholds.

MSapriori [17] is an extension of the existing association rule model that allows user to specify multiple minimum support thresholds in order to reflect different frequencies or natures of the items in the database. So the user can specify a different minimum support for each item (MIS) value. By providing different MIS values for different items, the user will effectively describe different support requirements for different rules. Let MIS (i) denote the MIS value of item i. The minimum support of a rule R is the lowest MIS value among the other items in the rule. Such that, a rule R, a_1 , a_2 , ……, a_r where a_j belongs to I , satisfies its minimum support if the rule's actual support is greater than or equal: min(MIS(a_1), MIS(a_1),……, MIS(a_r)) [17]

Example in [17]: Consider the following items in a database, *bread*, *shoes*, *clothes*.

The user-specified MIS values are as follows:

MIS(*bread*) = 2% , MIS(*shoes*) = 0.1%, MIS(*clothes*) = 0.2%

The following rule does not satisfy its minimum support:

clothes → *bread* [sup = 0.15%, conf = 70%]

because min(MIS(*bread*), MIS(*clothes*)) = 0.2%.

The following rule satisfies its minimum support:

clothes → *shoes* [sup = 0.15%, conf = 70%]

because min(MIS(*clothes*), MIS(*shoes*)) = 0.1% [17]. After the application of MSapriori algorithm [17], all frequent itemsets are found but the supports of some subsets may still be unknown. So, to generate association rules, a post processing phase is needed to find the supports of all subsets of frequent itemsets. This process is time consuming because we need another scan for the database to compute the supports of all subsets of frequent itemsets. To overcome both problems, a multiple item support tree (MIS-tree) is proposed which is an extension of the FP-tree structure, the CFP-Growth algorithm was developed to mine the complete set of frequent patterns with multiple minimum items supports [2].

The CFP-growth algorithm has the following three steps:

- Construction of a tree structure, called **MIS-tree**, using every item in the database.
- Generating compact MIS-tree by **pruning** the items from MIS-tree that cannot generate any frequent pattern.
- Mining compact MIS-tree using **conditional pattern bases** until all frequent patterns are generated.

Below is an example for CFP- Growth algorithm [2].

Table 2.1 shows the items sorted the right most column in descending order of their MIS values as given in Table 2.2:

Table 2.1. Transaction database D.

TID	Item bought	Item bought (ordered)
100	d, c, a, f	a, c, d, f
200	g, c, a, f, e	a, c, e, f, g
300	b, a, c, f, h	a, b, c, f, h
400	g, b, f	b, f, g
500	b, c	b, c

Table 2.2. Multiple minimum supports (MIS) values of each item in D.

Item	A	b	c	D	e	f	G	h
MIS value	80%(4)	80%(4)	80%(4)	60%(3)	60%(3)	40%(2)	40%(2)	40%(2)

The incompact MIS-tree is built using every item in the database D. The result MIS-tree is illustrated in Figure 2.2.

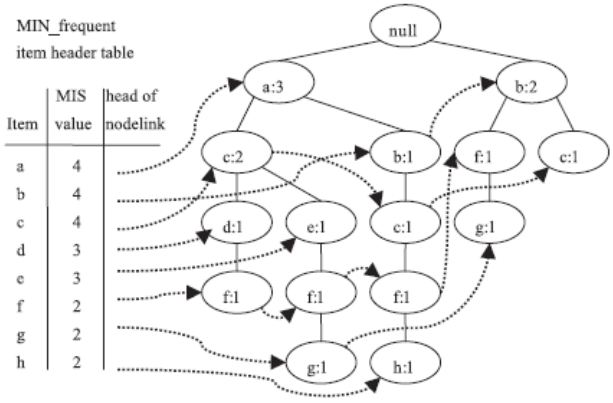


Figure 2.2. The incompact MIS-tree (before MIS_Pruning and MIS_Merge).

- Those items with supports no less than MIN = 2 (all items in F) in our MIS-tree should be retained. While these items {h, d, e} will be deleted because their support is less than MIN = 2. This process is called the pruning process.
- After removing these nodes, the remaining nodes in the MIS-tree may contain child nodes carrying the same item-name. MIS_Pruning process of MIS-tree is shown in Figure 2.3.

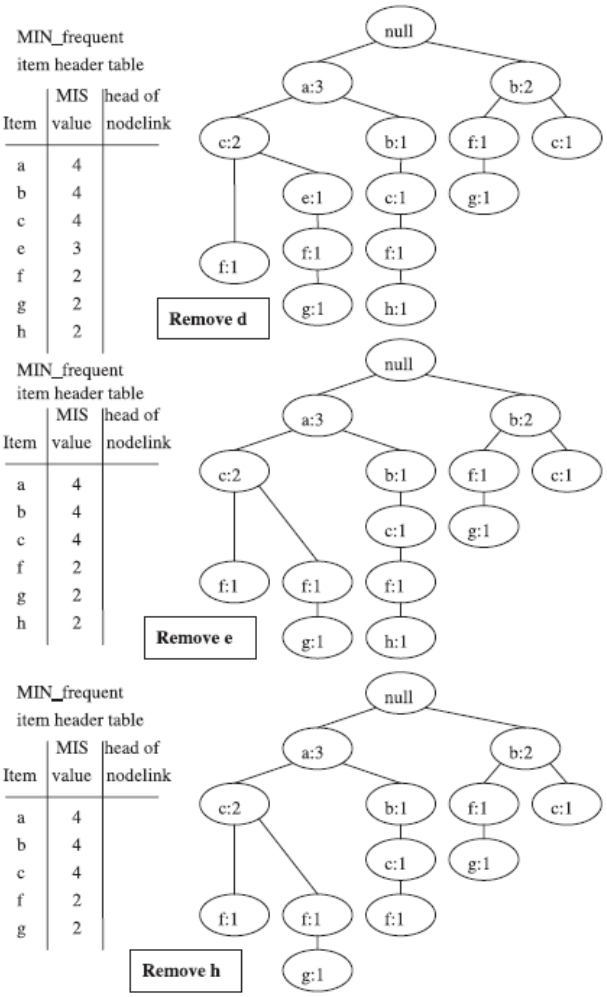


Figure 2.3. MIS_Pruning process of MIS-tree.

- For the sake of compactness, we traverse the MIS-tree and find that node (c:2) has two child nodes carrying the same item-name f.
- These two nodes are merged into a single node with item-name=f, and its count is set as the sum of counts of these two nodes.
- MIS_Merge process of MIS-tree after removing items d, e and h is shown in Figure 2.4.

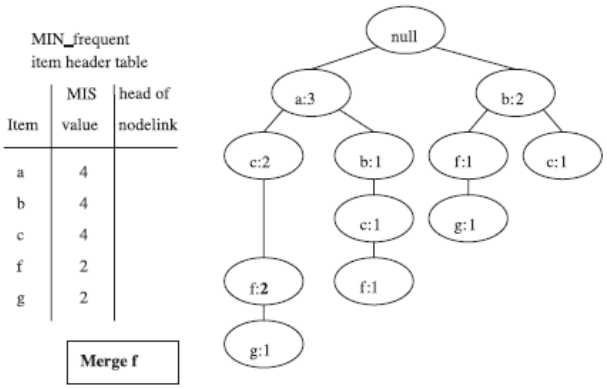


Figure 2.4. MIS_Merge process of MIS-tree.

The final result complete and compact MIS-tree is presented in Figure 2.5.

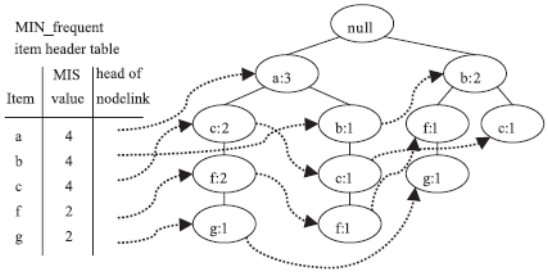


Figure 2.5. The complete and compact MIS-tree.

The CFP-Growth algorithm is examined by starting from the bottom of the header table

- To build a conditional pattern base and conditional MIS-tree for item g.
- There are two paths in the MIS-tree: (a:3, c:2, f:2, g:1) and (b:2, f:1, g:1).
  To build the conditional pattern base and conditional MIS-tree for g:
- The node g is excluded in these two paths, so (a:1, c:1, f:1) and (b:1, f:1) are the g's Conditional pattern base. The g's conditional MIS-tree is shown in Figure 2.6.

Table 2.3. Conditional pattern base.

Item	MIS	Conditional pattern base
g	2	{(a:1, c:1, f:1), (b:1, f:1)}
f	2	{(a:1, c:1),(a:1, b:1, c:1),(b:1)}
c	4	{(a:2), (a:1, b:1), (b:1)}
b	4	{(a:1)}
a	4	ϕ

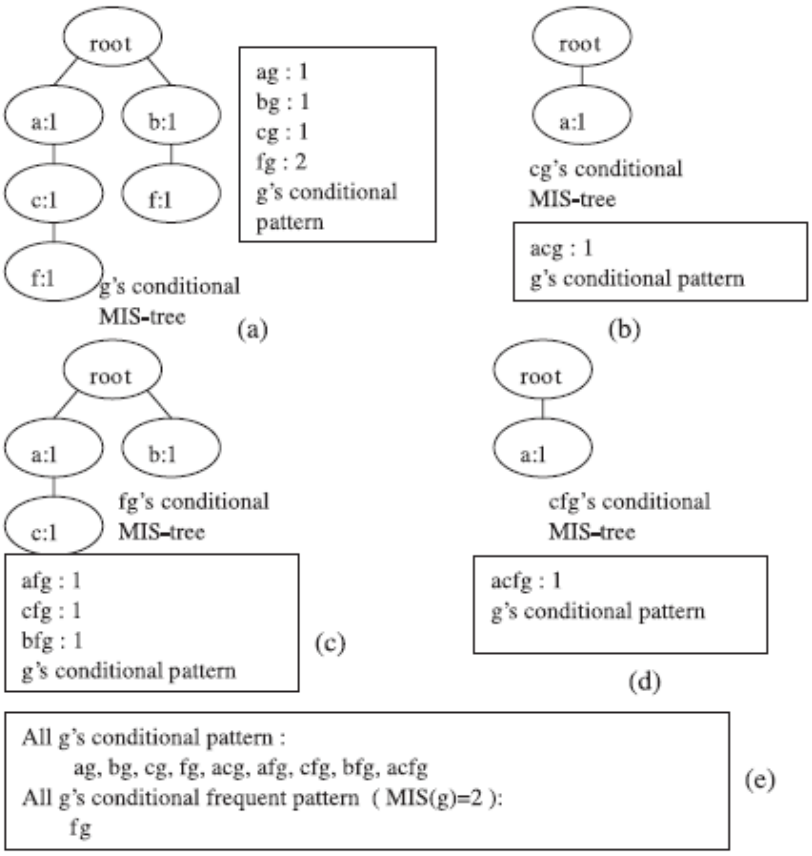


Figure 2.6. g's conditional MIS-tree.

- All the frequent pattern must be greater than or equal MIS (g).

By repeating this process for other items we will get the conditional patterns and conditional frequent patterns in Table 2.4.

Table 2.4. All Conditional patterns and conditional frequent patterns.

Item	Conditional patterns	Conditional frequent patterns
g	ag, bg, cg, fg, acg, afg, cfg, bfg, acfg	fg
f	af, bf, cf, abf, acf, bcf, abcf	af, bf, cf, abf, acf
c	ac, bc, abc	ϕ
b	ab	ϕ

An efficient CFP-Growth algorithm is proposed in [3] by proposing novel pruning techniques to Reduce Search Space which is called CFP-Growth++ algorithm. The CFP-growth++ algorithm is an improvement over CFP-growth algorithm. The differences between CFP-growth and CFP-growth++ are as follows:

- The CFP-Growth++ employs a better criterion to identify the items that cannot generate any frequent pattern. This criterion enables CFP-Growth++ to construct compact MIS-tree with only those items that can generate frequent patterns.
- The CFP-Growth++ will not search for frequent patterns until the conditional pattern base of a suffix pattern is empty. Instead, it tries to identify which suffix patterns can generate frequent patterns at higher order and perform search only in them [3].

- The steps involved in CFP-Growth++ are as follows;

  i. Construction of MIS-tree

  ii. Generating compact MIS-tree

  iii. Mining frequent patterns from the compact MIS-tree.

More explanation and a complete example about CFP-Growth++ are illustrated in Chapter 3.

MISFP-Growth [36], is an extended version of FP-Growth [14]. It is similar to FP-Growth with slight differences. The main differences between MISFP-Growth and FP-Growth are as follows;

1. FP-Growth is based on single threshold whereas MISFP-Growth is based on multiple item support thresholds.

2. Items in FP-Growth method are arranged in descending order in terms of their actual support whereas in MISFP-Growth items are sorted in descending order in terms of their support threshold values.

MISFP-Growth requires the following essential steps:

1. Scan database DB once to find out the actual support of each item.

2. Find the lowest minimum support threshold (MIN-MIS) among all items in database.

3. Scan DB once again to collect items that satisfy MIN-MIS in each transaction, sort them in the descending order of their predefined MIS and insert these items into the MISFP-Tree. If the appropriate node of an item exists, its count is increased by one. Otherwise, a new node is inserted in the MISFP-Tree.

4. Create MIN-MIS-frequent header table of MISFP-Tree, that holds items with support no less than MIN-MIS in descending order of MIS values of items. It consists of item-name, MIS of item and the head of nodelink that point to item's occurrences in the MISFP-Tree. Nodes that have the same itemname are linked in sequence to simplify tree traversal.

5. Build the conditional pattern base and the conditional MISFP-Tree of each suffix item whose support is greater than or equal to its predefined MIS. These two data structures represent the knowledge extracted from MISFPTree.

The experimental results indicate that MISFP-Growth performs better than CFP-Growth++ in term of both runtime and memory consumption.

All of the above algorithms ignore the dynamicity of the databases. However, transactional databases are dynamic in general. When new transactions arrive or some transactions are deleted from the database, these algorithms should be re-run in order to find the current frequent itemsets. Dynamic frequent itemset mining is the solution for that problem.

2.3 Dynamic Association Rule Mining Algorithms

The first group of incremental itemset mining algorithms are Apriori based [6] and [7]. Fast Update (FUP) Algorithm is the first algorithm proposed for incremental mining of frequent itemsets. It handles the databases with transaction insertion. The main working principle of this algorithm can be summarized in two steps. In the first step only new transactions are scanned to generate 1-itemsets. In the second step these itemsets are compared with the previous ones and all frequent itemsets of the same size are discovered iteratively. There are four possible cases in this algorithm when new transactions are added:

- Case 1: If the itemset is frequent both in the original database and the new transactions,
  the itemset is always frequent.
- Case 2: If the itemset is frequent in the original database but infrequent in the new transactions, the frequency of the itemset is determined from the existing information.
- Case 3: If the itemset is infrequent in the original database but frequent in the new transactions, the original database should be scanned in order to determine frequent itemsets.
- Case 4: If the itemset is infrequent both in the original database and the new transactions, the itemset is always infrequent.

The original database should be scanned only in Case 3. In the first iteration, new transactions are scanned. If the itemset is frequent in the original database, the support count is calculated by adding the supports in the original database and the new transactions.

This support count is compared with the support threshold of the updated database and if it does not satisfy the support threshold, the item is accepted as a loser and is pruned. Otherwise, when the itemset satisfies the support threshold, it remains to be frequent in the updated database. If the itemset is not frequent in the original database, it is a potential candidate set. If its support count fails to satisfy the minimum support threshold in the new transactions, the item is pruned. Otherwise, original database is scanned in order to determine its frequency. FUP significantly reduces the

number of candidate sets generated and is found to be 3 to 7 times faster than re-running Apriori for small support threshold. For larger support, FUP still outperforms [6].

FUP2 copes with both insertion and deletion of transactions, was proposed by [7]. The algorithm is an extended version of FUP and it is equivalent to FUP in the insertion case. Previous mining results are used in order to find frequent itemsets in the insertion case and in the deletion case as well. The frequent *k*-itemsets from previous mining results are used in order to divide the candidate set C_k into P_k and Q_k where P_k is the set of candidate itemsets which have been frequent previously and Q_k is the set of candidate itemsets, which have been infrequent before. The support counts of any candidate item in P_k are known from previous mining result, so scanning only deleted and inserted transactions is enough to update the support counts of candidates in P_k. The main working principle can be summarized in two steps. First, the deleted transactions are scanned so some candidate items can be deleted from P_k. On the other hand, the support counts of itemsets in Q_k are unknown because they have been infrequent. However, when an itemset in Q_k is frequent in the deletions, it must be infrequent in the updated database. Second, the inserted transactions are scanned. The insertion case is the same as FUP.

The second group of incremental itemset mining algorithms are without Candidate Generation like TIARM [4], IULFP [9] and Dynamic Matrix Apriori [10, 11]. TIARM (Tree-based Incremental Association Rule Mining) algorithm [4], is an extension of FP-growth algorithm and mines frequent pattern without candidate generation with different supports. TIARM is capable of handling transaction insertions as well as deletions, and achieves this by using a new data structure called INC-Tree (INCremental Tree), which is mainly intended to improve storage compression of FP-tree. Figure 2.7, Initially, INC-Tree is empty with a null root node. Then, transactions except the first one are preprocessed by sorting its items according to the item's appearance order in the database and also based on previously inserted transactions. In the next step, when transactions are inserted, the support count of the items is also updated.

TID	Original database	Sorted Database
1	E, D,A,C,B	E, D,A,C,B
2	F,G,B,C	C,B,F,G
3	B,F,D,E	E,D,B,F

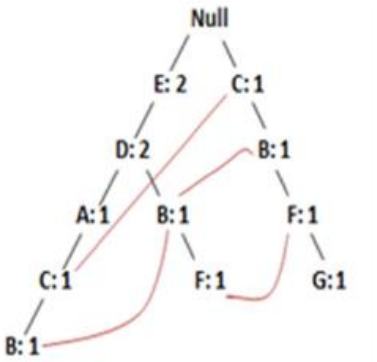


Figure 2.7. INC-Tree Construction.

Association rule mining is a two-step process:

• Frequent itemsets generation i.e. all the itemsets having support greater than the user specified minimum support.

• Frequent itemsets generated in the step 1 will be used to generate association rules that satisfy user specified minimum support.

Since association rules are generated directly from frequent itemsets, each rule automatically satisfies minimum support.

After constructing INC-Tree, it can be used for mining frequent patterns**.** First discard all the items from INC-Tree which are not satisfying user given min support. The sort-list is updated by removing all infrequent items one after another and at the same time the tree is pruned by deleting all nodes representing that item.

After building INC-Tree, new transactions can be added to and deleted from it. Lastly, mining of frequent patterns is done as in FP-Growth, with even different support values support values without rebuild the tree. TIARM follows divide and conquer method to generate frequent patterns without generating candidate itemsets as in FP-growth.

Steps of TIARM algorithm are:

```
TIARM (INC-Tree, min_supp)
If INC-Tree contains a single path P, then
For each combination (denoted as b) of the nodes in the
    path P,
    then
    Generate pattern b + a with support = min_supp of
    nodes in b
Else for each a in the header of tree, do
{
```

```
    Generate pattern b = a + a with support = a.support;
    Construct
    (1) b's conditional pattern base and
    (2) b's conditional INC-Tree Treeb
    If Treeb is not empty, then
    Call TIARM(INC-Tree, b);
}
```

IULFP (Incremental Updating algorithm based on LFP-tree) [9] uses a different data structure known as LFP-tree (Layered Frequent Pattern tree) to maintain frequent itemsets whenever the database is updated. As shown in Figure 2.7, LFP-tree is built by scanning the database and getting 1-itemsets and their frequency, which forms the first level of the tree. Every itemset in each level is represented as 3-tuple < a, v, t >, where "a" denotes the itemset, "v" denotes its frequency and "t" is a Boolean value which is either 1 or 0 to indicate whether the itemset is frequent or not, respectively. For the case of frequent 2-itemsets, they are generated from level 1 itemsets and linked to level 2 and the remaining frequent k -itemsets are linked to level k

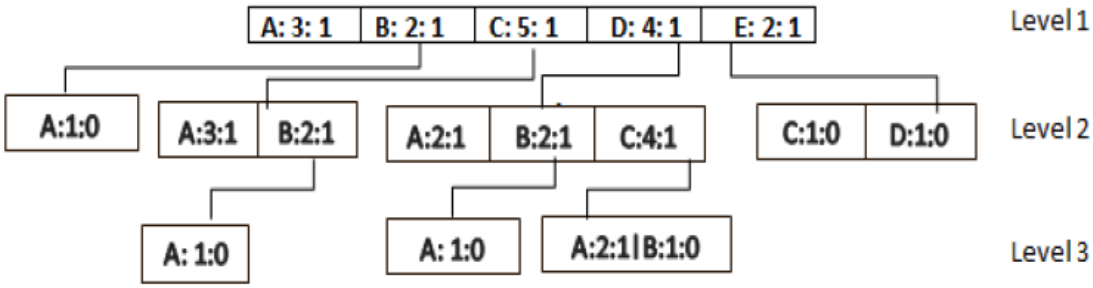


Figure 2.8. LFP- Tree Construction.

When transactions arrive, they are scanned once to get the support count of the itemsets. The itemset values for support in level 1 are updated and then determined if they are frequent or not. For the remainder of the levels, potential k-frequent itemsets are determined from the previous level, and later updated in the LFP-tree. Lastly, all frequent patterns are generated by inspecting each conditional pattern in the tree. From the comparison tests in [9], IULFP has a mining time of only 69% of FUP.

Unlike other dynamic rule mining algorithms that eliminate candidate generation, by representing the signature of database in a tree data structure, DMA (Dynamic Matrix Apriori) [10, 11] uses a matrix and vector as its data structures. DMA is built with the dynamic aspect in mind. This involves storing all the itemsets irrespective of their support in the matrix.

All of the above algorithms are either handle static with single and multiple support or dynamic with single support ignoring the dynamicity of the databases with multiple support thresholds. However, multiple support is very effective to solve the rare item problem. The transactional databases are dynamic in general. When new transactions arrive or some transactions are deleted from the database, these algorithms should be re-run in order to find the current frequent itemsets. So a solution for the problem of dynamic frequent itemset with multiple support mining is needed.

2.4 Dynamic Association Rule Mining with Multiple Support Algorithms

In [33] algorithms are developed for maintenance of the MIS tree after support tuning and incremental update of database without rescanning database. The maintenance algorithm is implemented for MS tuning violating the restriction used in the existing maintenance method. This algorithm is based on the CFP-Growth algorithm. The results indicate that it performs better than the existing approach (CFP-Growth). The algorithm is examined for the incremental approach. It performs faster than the method of reconstructing the MIS tree. In short, this research has two main results. First, it solved the problem occurred in the MIS tree algorithm in [2]. That it cannot tune supports of each item with the full flexibility without rescanning the database again. Second, it developed an efficient maintenance algorithm after incremental update that performs better than reconstruction MIS tree algorithm after update of database [33].

Actually this paper [33] have the same title as our thesis but the contents are different and it is based on different algorithm, the main differences are as follows:

- The algorithm in [33] is a CFP-Growth based. But our proposed algorithms are CFP-Growth++ based, while CFP-Growth ++ algorithm is a development of the CFP-Growth, and it performs better than CFP-Growth.

- The algorithms in [33] handle the maintenance and MS tuning mechanism in the based algorithm supposing the case of changing the MIS values of the items after building the tree but we assume that the MIS values do not change after building the tree.
- We build the static version of Dynamic MIS1 algorithm, but the algorithm in [33] is based on the static version of CFP-Growth.
- For the update purpose; our algorithms handle increments with additions, additions with new items and deletion. While the algorithm in [33] handle increments with additions.

Next chapter proposes our two algorithms which are Dynamic MIS1 and Dynamic MIS2 Algorithms. First, the base algorithm CFP-Growth++ is explained, and then the Dynamic MIS1and Dynamic MIS2 Algorithms are introduced with several examples. These examples show how each proposed algorithm handles the base tree building, additions, additions with new items and finally they demonstrate how the deletion is handled.

CHAPTER 3

DYNAMIC MINING UNDER MULTIPLE SUPPORT THRESHOLDS ALGORITHMS

There are several outstanding algorithms for the problem of itemset mining under single user given support value. However the major drawback of all these algorithms is their dependence on single user given support value. Single support is not enough because it may cause rare item problem [18]. So recently some algorithms are introduced for the problem of itemset mining under multiple support threshold.

When new transactions arrive or some transactions are deleted from the database, the problem of repeating the entire process of mining from the beginning occurs. Several research works have developed feasible algorithms for deriving precise association rules efficiently and effectively in such dynamic databases but they are devised for single support threshold.

The focus of this thesis is on dynamic update problem of frequent itemsets under multiple support thresholds; the challenge is to propose a solution that combines the two problems to mine the frequent itemsets under multiple support thresholds dynamically. In this study, two new dynamic itemset mining under multiple support thresholds algorithms which are called (Dynamic MIS1 and Dynamic MIS2) are introduced and explained, which are 1) tree based, 2) have one scan for the databases, 3) avoid the candidate generation problem, 4) they handle increments with additions, additions with new items and deletions.

This chapter is divided into three subsections, in the first the motivating example dataset. In the second subsection, the base algorithm CFP-Growth++ is explained. In the third subsection, our proposed algorithms Dynamic MIS1and Dynamic MIS2 are introduced with several examples. These examples show how each proposed algorithm handles the base tree building, additions, additions with new items and finally they demonstrate how the deletion is handled.

3.1 Motivating Example

Throughout the text, we use the following example presented in two tables. Table 3.1 [2] shows a sample database D that consists of five transactions. Table 3.2 on the other hand, shows the user given multiple item support (MIS) of each item in decreasing order. Last row of Table 3.2 shows actual support of each item in the database D. In the right most column of Table 3.1, items in the transactions are in an order of support values as given in Table 3.2 [2].

Table 3.1. Transaction database D.

TID	Item bought	Item bought (ordered)
100	D, C,A, F	A, C, D, F
200	G, C, A, F, E	A, C, E, F, G
300	B, A, C, F, H	A, B, C, F, H
400	G, B, F	B, F, G
500	B, C	B, C

Table 3.2. MIS and actual support of each item in D.

Item	A	B	C	D	E	F	G	H
MIS value	80%	80%	80%	60%	60%	40%	40%	40%
Actual Support	60%	60%	80%	20%	20%	80%	40%	20%

3.2 Itemset Mining under Multiple Support Thresholds

There are several algorithms for the problem of itemset mining under single user given support value. Apriori [13], FP-Growth [14] and Matrix Apriori [15]. However major drawback of all these algorithms is their dependence on single user given support value. Single support is not enough because it may cause rare item problem [18]. So recently some algorithms like MSapriori[17] , CFP-Growth[2] , CFP-Growth++[3] and MISFP-Growth [36] are introduced for the problem of itemset mining under multiple support threshold.

In this section we will explain the CFP-Growth++ algorithm since we use this algorithm in our proposed dynamic approaches which are explained in the following section. CFP-Growth++ algorithm has following steps;

Step 1: Construction of MIS-tree

Step 2: Generating compact MIS-tree

Step 3: Mining frequent patterns from the compact MIS-tree.

In our two algorithms we apply Step 1 dynamically. Our second algorithm *Dynamic MIS*2 follows Step 2 and Step 3 in the same way as CFP-Growth++. On the other hand, *Dynamic MIS*1 algorithm skips Step 2 and apply Step 3 of CFP-Growth++ for only the items that exist in the primary header table.

Step 1 Construction of MIS-tree: The MIS-tree consists of two components: MIS-list and prefix-tree. The MIS-list is a list having three fields {item name (item), support (S) and minimum item support (MIS)}. The structure of the prefix-tree in MIS-tree is same as that in FP-tree [2] but items here sorted by MIS. The items are sorted in descending order of their MIS values and items are inserted into the MIS-list with support equal to zero. MIS-tree is then created like FP-growth given in [14].

Step 2 Construction of compact MIS-tree: A method to prune such items from the MIS-tree is as follows:

i. Starting from the last item of the MIS-list, the items that have support less than their respective MIS value are pruned.

ii. Once the frequent item is found, its MIS value is chosen as the MIN MIS (minimum MIS) value. Next , support of the remaining items in the MIS-list are compared with MIN MIS value, and those items that have support less than MIN MIS are pruned from the MIS-tree

iii. After tree-pruning, tree-merging process is carried out to merge the child nodes of a parent node that share a common item.

iv. The compact MIS-tree is generated after tree-pruning and tree-merging operations.

v. The process of infrequent leaf node pruning is carried on the compact MIS-tree to decrease its size.

Step 3 Mining frequent patterns from compact MIS-tree: Conditional minsup and conditional closure property are used for mining frequent patterns [3]. It is started by the lowest level items and repeated until all frequent patterns are generated.

Now let us demonstrate the flow of CFP-growth++ algorithm with our motivating example given in Table 3.1 and Table 3.2. The resultant incompact MIS-tree is given in Figure1. Its generation is same as the generation of FP-tree [2].

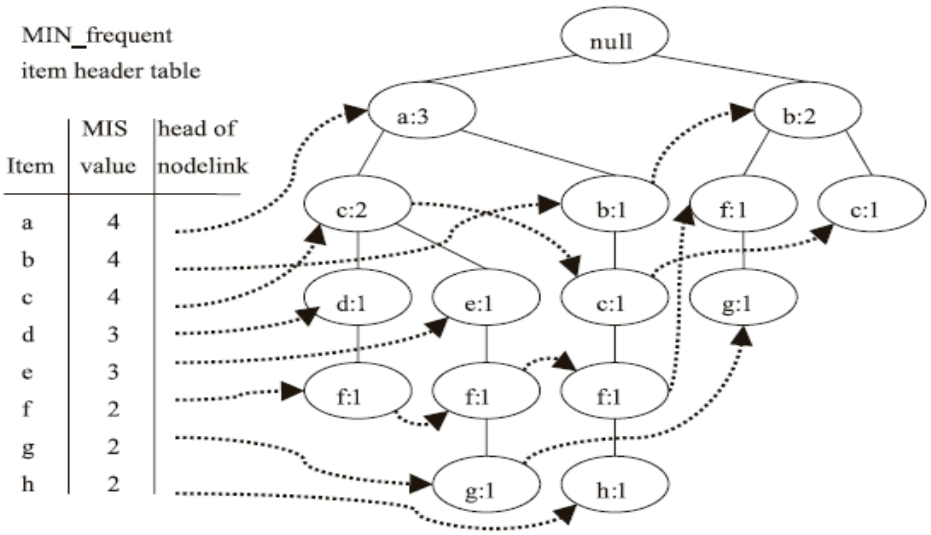


Figure 3.1. The incompact MIS tree.

The MIS-tree is constructed with every item in the transaction database. So, lowest multiple support (MIN MIS) and infrequent leaf node pruning are used in pruning step (Step 2) to decrease the search space. MIN MIS is 2(40%) from the MIS-tree. As a result, any item that has support less than 2 is discarded (i.e., D, H, E).The last item in the MIS-list is G that is frequent item. Hence, no tree-pruning operation is done for the item G. The tree-pruning operation ends as the supports of the remaining items in the MIS-tree are greater than MIN MIS = 2.

After tree-pruning, tree-merging process is carried out to merge the child nodes of a parent node that share a common item like FP-growth given in [2]. The result MIS-tree is called compact MIS-tree as given in Figure 3.2. The process of infrequent leaf node pruning is carried on the compact MIS-tree to decrease its size. The process is as follows. Among the remaining items in the MIS-list of MIS-tree, A and B are infrequent items (i.e., their support is less than the required minsup value). Therefore, using the node-links of A and B, we collect all the branches containing A or B. The branches containing A are {{A, C: , F , G : 1}, {A, B, C, F :1 }}. In these branches A is not leaf node so we cannot delete it. As the same way B is not a leaf node, so it will not be removed because the coming pattern contains B may be frequent.

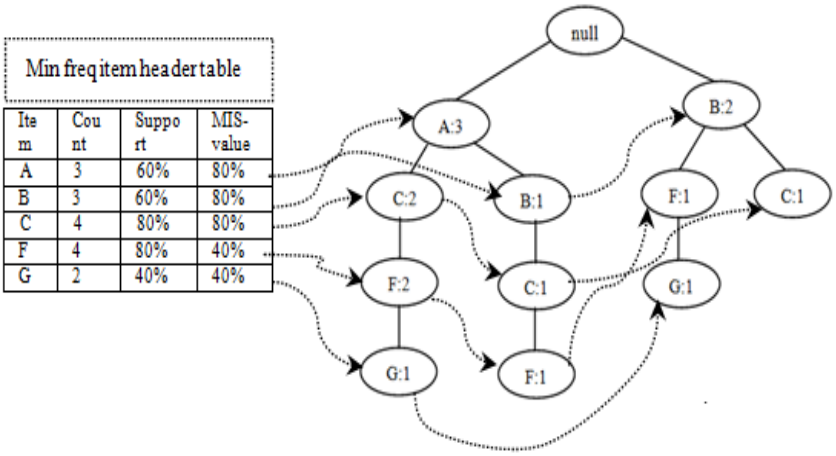


Figure 3.2. The compact MIS-tree after pruning and merging.

Now let us explain how mining frequent patterns are generated from compact MIS-tree is done. Consider the item G that has lowest MIS among all items in the compact MIS-tree. It occurs in 2 branches of compact MIS-tree. The branches are {A, C, F, G} and {B, F, G}. Considering G as a suffix pattern, its conditional prefix paths are {A, C, F} and {B, F} which form its conditional pattern base.

As the compact MIS-tree is constructed in MIS descending order of items, G (suffix item) will have lowest MIS value among all the items in its conditional pattern base. Therefore, using MIS value of the item g (i.e., 2) as the conditional minsup, conditional MIS-tree is generated with {F: 2} the items A, B and C are not included because the support counts are less than the specified conditional minsup value (i.e., conditional closure property).

The single path generates the frequent pattern {F, G: 2}.Similar process is repeated for other remaining items in the compact MIS-tree to discover the complete set of frequent patterns. The results are shown in Table 3.3.

Table 3.3. Conditional pattern base and frequent pattern mining results.

Suffix item	Conditional minsup	Conditional pattern base	Conditional MIS-tree	Frequent pattern
G	2	{ A, C , F :1}, { B , F : 1}	{F:2}	FG:2
F	2	{A,C :2},{A,B,C :1},{B :1 }	{A:2},{C:2},{B:2}, {AC:3}	AF:2,CF:2, BF:2, ACF:2
C	4	{A:2},{A,H:1} ,{B}	-	-
B	4	{A:1}	-	-
A	4	-	-	-

3.3 Dynamic Frequent Itemset Mining Algorithms

When the database is updated, the problem of repeating the whole process of mining from the beginning occurs. Several research works have developed feasible algorithms for deriving precise association rules efficiently and effectively in such dynamic databases. Such as FUP [6, 8], FUP2 [7], Prelarge trees [5], TIARM [4], IULFP [9],and DMA [15, 16]. However all these algorithms can cause rare item problem since they are designed for single support threshold.

In this chapter we introduce two new algorithms; *Dynamic MIS*1 and *Dynamic MIS*2 algorithms. They provide a solution to the dynamic itemset mining under multiple support thresholds problem. The previous works handle either the dynamic itemset mining with single support threshold or static with multiple support thresholds. Our two approaches take the advantages of the previous approaches and combine them to provide a solution to dynamic aspect of itemset mining problem under multiple support thresholds. Both approaches handle additions, additions with new items and deletions in increments. Our two algorithms use tree based data structure to minimize database scan. *Dynamic MIS*1 algorithm uses two header tables, while *Dynamic MIS2* uses one header table. Mining using CFP-Growth++ [3] is applied in both of them in mining frequent patterns from the tree. Compacting the tree before mining is required only for the second one.

3.3.1 Dynamic MIS1 Algorithm

This algorithm builds the first MIS-tree as in [2] from the original DB and divide the items between primary and secondary header tables according to their status, such that the primary header table contains the items which have support more than the MIN MIS value, and the secondary table contains the other items. Each time an incremental database d comes, it will be added to the existing tree, as a result; some items need to be added to the tables while others need to change their place between the two header tables. After that, mining is applied only for the items that exist in the primary header table.

3.3.1.1 Building MIS-tree

```
INPUT: Original database D, Minimum support thresholds of items MIS

OUTPUT: Sorted MIS values MIS_sorted, Multiple item support tree MIS-tree

BEGIN

1     Build sorted list MIS_sorted of the MIS values in decreasing
      order

2     Create the root of MIS-tree and label it as null

3     Create primary and secondary header tables

4     Insert items in the primary table with count 0

5     Scan D

6     FOR each transaction T in D do:

7       Sort all items in T according to MIS_sorted

8       Add T to the tree

9     END FOR

10    Calculate the support of items in D

11    Update the supports in the tables

12    Relocate items between the two tables

13    Return MIS_sorted, MIS-tree

END
```

Figure 3.3. MIS-tree builder algorithm.

In order to build the MIS-tree, the MIS-tree builder algorithm given in Figure 3.3 is used. First, the $\mathbf{MIS_{sorted}}$ list is created from the MIS values and ordered in decreasing order as indicated in Line 1 of Figure 3.3. After that primary header table (a table with columns: item's name, item's count, item's support, item's MIS value) and secondary header table (a table with columns: item's name, item's count, item's support, item's MIS value) are created (Line 3). Then the items ordered as $\mathbf{MIS_{sorted}}$ are inserted into the primary header table with item's count 0 as in Line 4. Database D is scanned, and the transactions are added to the tree (Lines 6-9). First, the items in the new transaction are sorted in decreasing order as the $\mathbf{MIS_{sorted}}$ list. Then transaction is added to the tree, such that transactions that share prefix with other transactions, these prefixes are incremented by 1, otherwise; new nodes will be created starting from the root node with item's count equal to 1. Update each item's count in this transaction by incrementing its count in the primary header table by 1. Then link the nodes of same item all through the tree and to the header table. This entire process is indicated in Lines 5-9.

The supports of all items in D are calculated then updated in the header table (Lines 10-11). Finally, the items are located in the two tables; such that; the items with support more than the MIN MIS value are inserted into the primary header table or else the items with support less than the MIN MIS value are inserted into the secondary header table, their node links are changed accordingly (Line 12) .

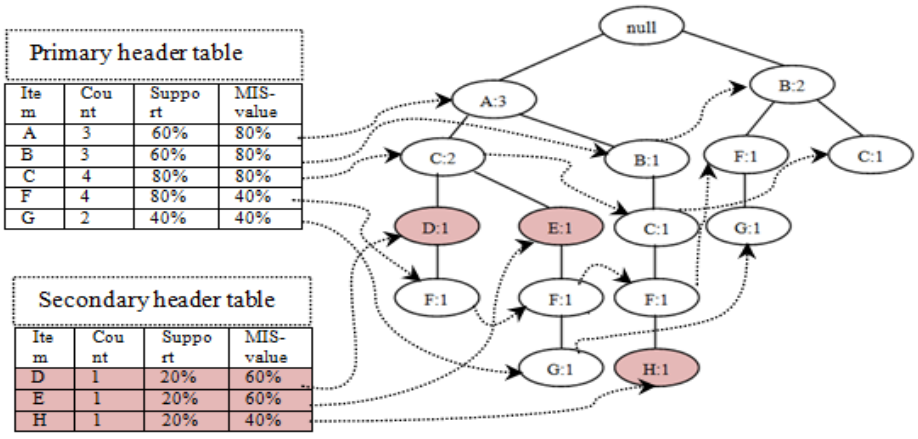


Figure 3.4. MIS-tree using MIS tree builder algorithm.

Let us explain MIS-tree builder algorithm with the example presented in Table 3.1 and Table 3.2. First, the $\mathbf{MIS_{sorted}}$ list is created from the MIS values in Table 3.1 and ordered in decreasing order. After that, the root node of the tree is created. Primary header table and secondary header tables are created as shown in Figure 3.4. Then the items ordered as $\mathbf{MIS_{sorted}}$ are inserted into the primary header table with item's count 0. The database D is scanned, and the transactions are added to the tree. First, the items in the new transaction are sorted in decreasing order according to $\mathbf{MIS_{sorted}}$ list as shown in the right most column of Table 3.1.Then transaction is added to the tree, such that transactions that share prefix with other transactions, these prefixes will be incremented by one, otherwise; new nodes will be created starting from the root node with item's count equal one. Update each item's count in this transaction by incrementing its count in the primary header table by 1. Then link the nodes of same item all through the tree and to the header table. The supports of all items in D are calculated then updated in the header table. Finally, the items are located in the two tables. Such that; the items with support more than the MIN MIS value (40%) are inserted into the primary header table, otherwise; the items with support less than the MIN MIS value (40%) are inserted in the secondary header table. Also the node links are arranged. The items' insertions into the tables are done with respect to the order of the sorted list.

3.3.1.2 Adding Increments

Let us explain the pseudo code of the update process for additions which is given in Figure 3.5. New transactions arrive, they are scanned to be added to the tree as follows. First, the items in the new transaction are sorted in decreasing order as $\mathbf{MIS_{sorted}}$ list. Then transaction is added to the tree, such that transactions that share prefix with other transactions, these prefixes will be incremented by 1, otherwise; new nodes will be created starting from the root node with item's count equal 1. Update each item's count in this transaction by incrementing its count in the primary header table by 1. Then link the nodes of same item all through the tree and to the header table. This entire process is indicated in Lines 1-5. The supports of all items in the whole database are calculated then updated in the header tables as in Lines 6-7. Finally, compare each item support with its MIN MIS value; the items which were in the primary table and

their supports become less than the MIN MIS value, they are transferred to the secondary header table. Otherwise; the items which are in the secondary header table and their supports are more than the MIN MIS value, they are transferred to the primary header table. The transfer operations are done with respect to the items' order sort list $\mathbf{MIS_{sorted}}$. This process indicated in Line 8.

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted,
incremental database d

OUTPUT:Dynamic MIS-tree

BEGIN

1     Scan d

2     FOR each transaction T in d do:

3       Sort all items in T according to MIS_sorted

4       Add T to the tree

5     END FOR

6     Calculate the support of items

7     Update the supports in the tables

8     Relocate the items between the primary and secondary header
      tables

9     Return Dynamic MIS-tree

END
```

Figure 3.5. Update process in Dynamic MIS1 for additions.

Table 3.4. The incremental database d.

TID	Item bought	Item bought (ordered)
1	C, B, H	B ,C, H
2	G, B, F	B ,F, G
3	C, D, H	C ,D, H

Let us give an example for the addition process. This example is based on the MIS-tree in Figure 3.4, its sort list on MIS values Table 3.2 and the incremental database given in Table 3.4. When new transactions arrive (Table 3.4), they are scanned to be added to the tree. First, the items in the new transaction are sorted in decreasing order of $\mathbf{MIS_{sorted}}$ list as in the right most column in table above. Then transactions are

added to the tree one by one as in Figure 3.6. Each item's count in this transaction is updated by incrementing its count in the primary header table by 1. Then link the nodes of same item all through the tree and to the header tables of the same figure. The supports of all items in the whole database are calculated then updated in the header tables. Finally, items are relocated between header tables, each item support is compared with its MIN MIS value. Here we note that the items (A and G) are transferred from the primary to secondary header table, because their supports become less than the MIN MIS value (40%). Note that when items (A and G) are inserted into the secondary table. Figure 3.6 shows the result *Dynamic MIS-tree*.

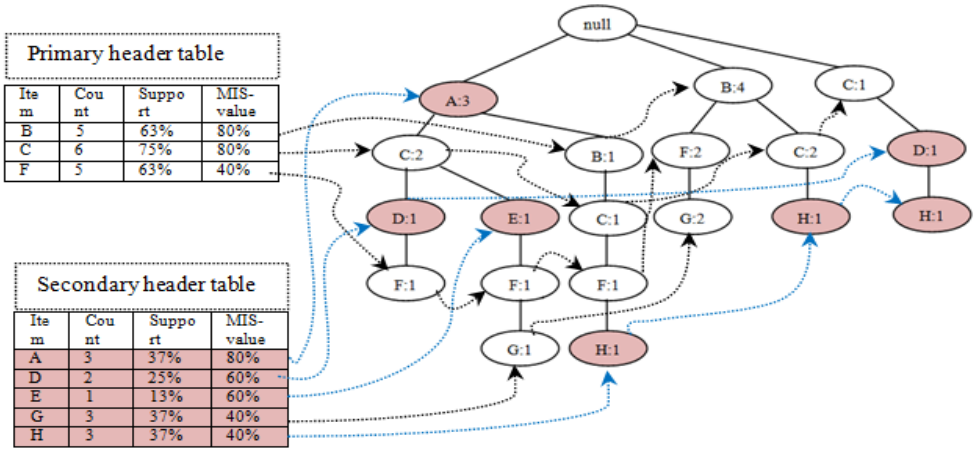


Figure 3.6. Dynamic MIS-tree after adding d.

Now the *Dynamic MIS-tree* is ready for mining using the modified CFP-Growth++ algorithm presented in the previous section for only the items which are in the primary header table. We use this algorithm because we want to apply mining for frequent items with multiple support thresholds.

3.3.1.3 Adding Increments with New Items

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted,
incremental database d, Minimum support threshold of new items MIS_new

OUTPUT: MIS_sorted, Dynamic MIS-tree

BEGIN

1     Build new sorted list MIS_sorted from MIS_sorted and MIS_new in
      decreasing order.

2     Insert the new items in the primary table with count 0

3     Scan d

4     FOR each transaction T in d do:

5       Sort all items in T according to MIS_sorted

6       Add T to the tree

7     END FOR

8     Calculate the support of all items

9     Update the supports in the tables

10    Relocate the items between the primary and secondary header
      tables

11    Return MIS_sorted, Dynamic MIS-tree
```

Figure 3.7. Dynamic MIS1 for additions with new items.

Let us explain the pseudo code of the update process for additions with new items which given in Figure 3.7. When new items appear, the $\mathbf{MIS_{sorted}}$ is updated by adding the new MIS values it in decreasing order as in Line 1. Then the new items in $\mathbf{MIS_{new}}$ are appended to the primary header table with item's count 0 and their MIS value as in Line 2. These two lines are the main difference between additions and additions with new items. The remaining steps of this process are similar to additions without new items previously.

Table 3.5. MIS values for new items in d.

Item	J	K	L
MIS value	70%	35%	30%

Table 3.6. The incremental database d with new items J, K, L.

TID	Item bought	Item bought (ordered)
1	C, B, K, J, H, L	B ,C, J, H, K, L
2	K , H	H, K
3	K, B, C	B , C, K

Table 3.7. MIS values of all items (old values and new values).

Item	A	B	C	J	D	E	F	G	H	K	L
MIS value	80%	80%	80%	70%	60%	60%	40%	40%	40%	35%	30%

Let us give an example for the addition with new items process. The input tree for this case is the MIS-tree in Figure 3.4. The first step is to combine the new MIS values in Table 3.5 with the MIS values of the old items in Table 3.2 to get Table 3.7. Items are arranged in decreasing order according to the MIS values. Then the new items in Table 3.6 are appended to the primary header table with item's count 0 and their MIS values. New transactions in d are scanned (Table 3.6) and then added to the tree as in Figure 8. Item counts are updated. From these counts, supports are calculated. According to the new items' supports, some items will be transferred from the primary header table to the secondary header table, and vice versa. In our example; item (G) is transferred from primary to secondary because its new support (25%) is less than the new MIN MIS value (30%). And item (H) is transferred from secondary to primary because it new support (37%) becomes higher than the MIN MIS value (30%). Also new items that appeared in d are added to the either one of the two tables according to their supports. It is important to notice that the transfer operations and new items' addition are done with taking into account the items' MIS order as in Table 3.7.Figure 3.8 shows the result MIS-tree after adding the three transactions.

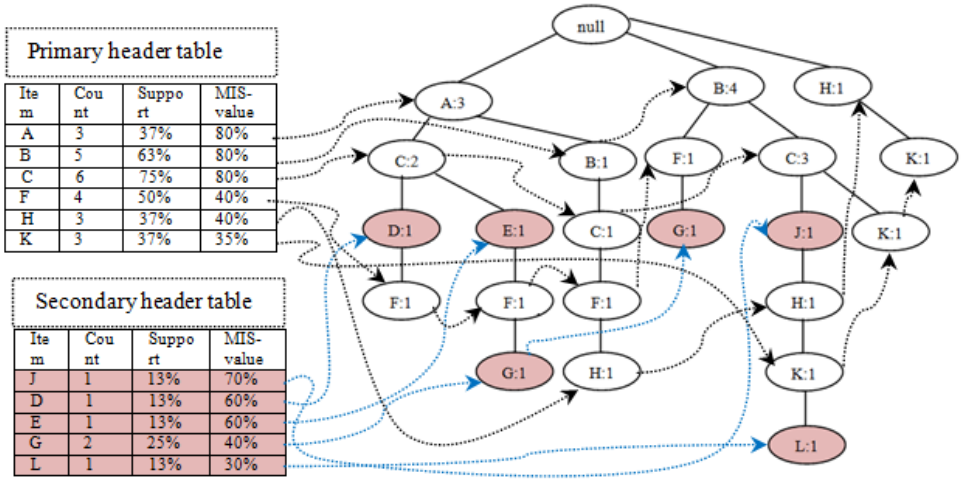


Figure 3.8. Dynamic MIS-tree after adding d with new items.

3.3.1.4 Adding Increments with Deletions

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted,
incremental database d

OUTPUT: Dynamic MIS-tree

BEGIN

1      Scan d

2      FOR each transaction T in d do:

3        Sort all items in T according to MIS_sorted

4        Delete T from the tree

5      END FOR

6      Calculate the support of all items

7      Update the supports

8      Relocate the items between the primary and secondary header
       tables

9      Return Dynamic MIS-tree

END
```

Figure 3.9. Update process in Dynamic MIS1 for deletions.

Let us explain the pseudo code of the update process for deletions which is given in Figure 3.9. When new transactions arrive, they are scanned to be deleted from

the tree. First, the items in the new transaction are sorted in decreasing order as $\mathbf{MIS_{sorted}}$ list. Then transaction is deleted from the tree, such that transactions that have count greater than 1are decremented by 1, otherwise; nodes are deleted. Each item's count is updated in this transaction by decrementing its count in the header table by 1. Then link the nodes of same item all through the tree and to the header table. This entire process is indicated in Lines 1-5. The supports of all items in the whole database are calculated then updated in the header tables (Lines 6-7). Finally, each item support is compared with its MIN MIS value; the items which are in the primary table and their supports become less than the MIN MIS value, they will be transferred to the secondary header table. Otherwise; the items which was in the secondary header table and their supports become more than the min MIS value, they will be transferred to the primary header table. The transfer operations are done with respect to the items' order sort list $\mathbf{MIS_{sorted}}$. This process indicated in Line 8.

Table 3.8. Transactional database d with deletions.

TID	Item bought	Item bought (ordered)
100	D,C,A,F	A ,C, D, F
400	B,F,G	B, F,G

Table 3.8 describes two transactions to be deleted from the original tree. The rightmost column of it lists all the items in each transaction following this order according to their MIS values in decreasing order as Table 3.2.

Here we will apply the deletion example on the tree of Figure 3.4. The new transactions in d in Table 3.8 are scanned and then deleted from the tree as in Figure 3.10. Some items' counts are decremented. From these counts supports are calculated and updated in the tables of tree. According to the new items' supports, some items will be transferred from the primary header table to the secondary header table, and vice versa. In our example; the support of item (G) is 33.3%, which is less than the MIN MIS value(40%). So it is removed from the primary header table and inserted into the secondary header table taking into account the order of items.

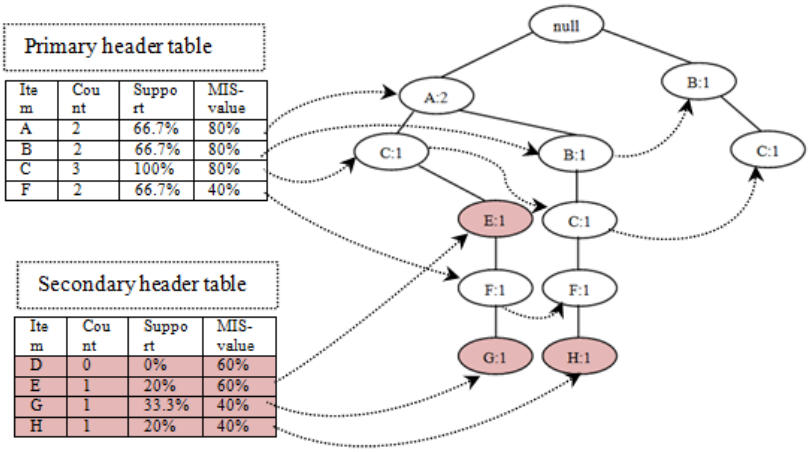


Figure 3.10. Dynamic MIS-tree after deletions.

After applying the decrement on the items of the deletion database, the nodes with count 1 are deleted from the tree, but their records are kept in its specified table, and other nodes are decremented by 1 for each deleted transaction. The result *Dynamic MIS-tree* is shown in Figure 3.10.

Now the *Dynamic MIS-tree* contains the complete information for frequent pattern mining with multiple MS. So we can do mining for the tree by the same way as CFP-Growth++ algorithm [3] for only the items that exist in the primary header table. When we apply mining for each item we skip the items that exist in secondary header table.

3.3.2 Dynamic MIS2 Algorithm

This algorithm builds the first MIS-tree as in [2] from the original DB, and each time an incremental database d is added, it will be added to the existing tree. This tree is reserved and pruning is applied as a preparation phase before mining. Mining will be applied using CFP-Growth++ in [3]. Once Tree is built, new transaction can be added to and deleted from MIS-tree, frequent patterns can be mined with multiple support values without rebuilding the tree.

3.3.2.1 Building MIS Tree

```
INPUT: Original database D, Minimum support thresholds of items MIS

OUTPUT: Sorted MIS values MIS_sorted, Multiple item support tree MIS-tree

BEGIN

1     Build sorted list MIS_sorted of the MIS values in decreasing
      order.

2     Create the root of MIS-tree and label it as null

3     Create Min frequent item header table

4     Insert sorted items in the table with count 0 for each

5     Scan D

6     FOR each transaction T in D do:

7       Sort all items in T according to MIS_sorted

8       Add T to the tree

9     END

10    Calculate the support of items in D

11    Update the supports in the header table

12    Return MIS_sorted, MIS-tree

END
```

Figure 3.11. MIS-tree builder algorithm.

In order to build the first MIS-tree, MIS2-tree builder algorithm in Figure 3.11. is used, First, $\mathbf{MIS_{sorted}}$ list is created from the MIS values and ordered in decreasing order as indicated in Line 1. After that Min frequent item header table (a table with columns: item's name, item's count, item's support, item's MIS value). Then insert the ordered items of the $\mathbf{MIS_{sorted}}$ into this table with item's count 0 as in Lines 3-4. Then the database D is scanned, and the transactions are added to the tree as follows. First, the items in the new transaction are sorted in decreasing order as the $\mathbf{MIS_{sorted}}$ list. Then transaction is added to the tree, such that transactions that share prefix with other transactions, these prefixes will be incremented by one, otherwise; new nodes will be created starting from the root node with item's count equal one. Update each item's count in this transaction by incrementing its count in the primary header table by 1. This

entire process is indicated in Lines 5-9. The supports of all items in D are calculated then updated in the header table as in Lines 10-11.

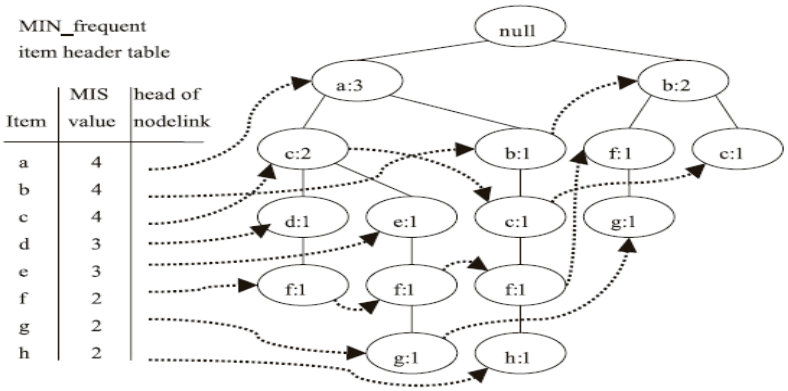


Figure 3.12. The incompact MIS-tree.

Based on Tables 3.1 and Table 3.2, we try to explain the flow of MIS-tree builder algorithm. $\mathbf{MIS}_{\mathtt{sorted}}$ list is created from the MIS values of Table 3.2and ordered in decreasing order as indicated. After that create Min frequent item header table as in Figure12. Then insert the ordered items of the $\mathbf{MIS}_{\mathtt{sorted}}$ in this table with item's count 0. The database D is scanned, and the transactions are added to the-tree. The items counts and supports are updated. Figure12 shows the incompact tree from this example. After that mining is applied using CFP-Growth++ algorithm.

3.3.2.2 Adding Increments

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted, incremental database d

OUTPUT: Dynamic MIS-tree

BEGIN

1     Scan d

2     FOR each transaction T in d do:

3       Sort all items in T according to MIS_sorted

4       Add T to the tree

5     END FOR

6     Calculate the support of all items

7     Update the supports in the Min frequent item header table

8     Return Dynamic MIS-tree

END
```

Figure 3.13. Update process in Dynamic MIS2 for additions.

We explain the pseudo code of the update process for additions that is given in Figure 3.13. When new transactions arrive the database d is scanned, and the transactions are added to the tree as follows. The items in the new transaction are sorted in decreasing order as the $\mathbf{MIS_{sorted}}$ list. Then transaction is added to the tree, such that transactions that share prefix with other transactions, these prefixes will be incremented by one, otherwise; new nodes will be created starting from the root node with item's count equal one. Each item's count in this transaction is updated by incrementing its count in the primary header table by 1. Then the links the nodes of same item all through the tree and to the header table are updated. This entire process is indicated in Lines 1-5. The supports of all items in the whole database are calculated then updated in the Min frequent item header table as in Lines 6-7.

Table 3.9. Transactional database d.

TID	Item bought	Item bought (ordered)
1	C, B, H	B, C, H
2	G, B, F	B , F, G
3	C, D, H	C, D, H

Let us give an example for the addition process. Table 3.9 shows incremental database d that consists of three transactions, and the right most column shows these transactions ordered according to the sort list in Table 3.2.

This example is based on the MIS- tree in Figure 3.12 and its sort list MIS values Table2.When new transactions arrive (Table 3.9), they are scanned to be added to the tree. The items in the new transaction are sorted in decreasing order as the $\mathbf{MIS}_{\mathtt{sorted}}$ list as in the right most column in table above. Then transactions are added to the tree one by one as in Figure14. Each item's count in this transaction is updated by incrementing its count in the primary header table by 1. Then link the nodes of same item all through the tree and to the header tables of the same figure. The supports of all items in the whole database are calculated then updated in the MIN-frequent item header table. Figure 3.14 shows the result *Dynamic MS-tree* after addition.

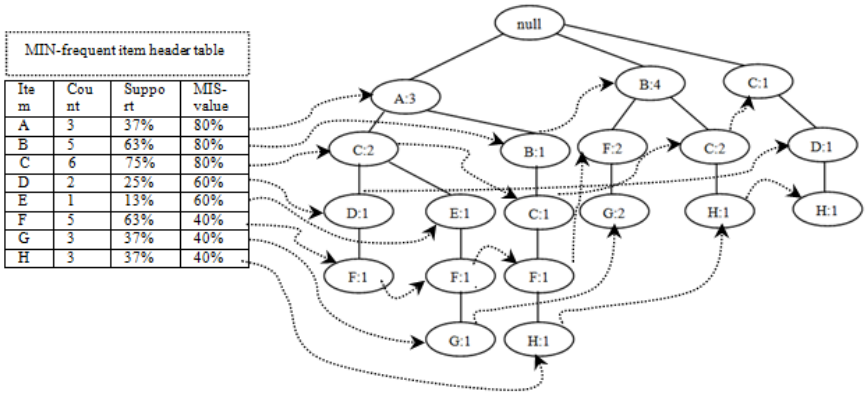


Figure 3.14. Dynamic MIS-tree after adding d.

Now the *Dynamic MIS-tree* is ready for mining using the second and third steps of CFP-Growth++ algorithm.

3.3.2.3 Adding Increments with New Items

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted,
incremental database d, Minimum support threshold of new items
MIS_new

OUTPUT: New Sorted MIS values MIS_sorted, Dynamic MIS-tree

BEGIN

1     Build new sorted list MIS_sorted from MIS_sorted and MIS_newin
      decreasing order.

2     Insert the new items in the Min frequent item header table
      with count 0 for each

3     Scan d

4     FOR each transaction T in d do:

5       Sort all items in T according to MIS_sorted

6       Add T to the tree

7     END FOR

8     Calculate the total support of all items

9     Update the supports in the Min frequent item header table

10    Return MIS_sorted, Dynamic MIS-tree

END
```

Figure 3.15. Update process in Dynamic MIS2 for additions with new items.

Let us explain the pseudo code of the update process for additions with new items which given in Figure 3.15. When new items appear, the $\mathbf{MIS_{sorted}}$ is updated by adding the new MIS values $\mathbf{MIS_{new}}$ to it then sorting the combination in decreasing order as in Line 1. Then the new items in $\mathbf{MIS_{new}}$ are inserted in the Min frequent item header table with item's count 0 and their MIS values, this insertion is done with taking into account the $\mathbf{MIS_{sorted}}$ order. This is indicated in Line 2. These two lines is the main difference between additions and additions with new items. The remaining steps of this process is similar to additions without new items that given in Figure 3.13.

Table 3.10. MIS values table for the new items in d.

Item	J	K	L
MIS value	70%	35%	30%

Table 3.11. Incremental database d with new items J, K, L.

TID	Item bought	Item bought (ordered)
1	C, B, K, J, H, L	B ,C, J, H, K, L
2	K , H	H, K
3	K, B, C	B , C, K

Table 3.12. MIS values of all items (old values and new values).

Item	A	B	C	J	D	E	F	G	H	K	L
MIS value	80%	80%	80%	70%	60%	60%	40%	40%	40%	35%	30%

The input tree for this case is the MIS-tree in Figure12. The first step is to combine the new MIS values in Table 3.2 with the MIS values of the old items in Table 3.10 to get Table 3.12 in non-increasing order of MIS values.

Then the new items in $\mathbf{MIS_{new}}$ are inserted in the Min frequent item header table with item's count 0 and their MIS values, this insertion is done with taking into account the $\mathbf{MIS_{sorted}}$ order. Now the increment d in Table 3.11 is scanned, and then sorted according to Table 3.12. After that; the transactions are added to the tree, then the item's counts and node links are updated as shown in Figure 3.16. From these counts the supports are calculated and updated in the header table.

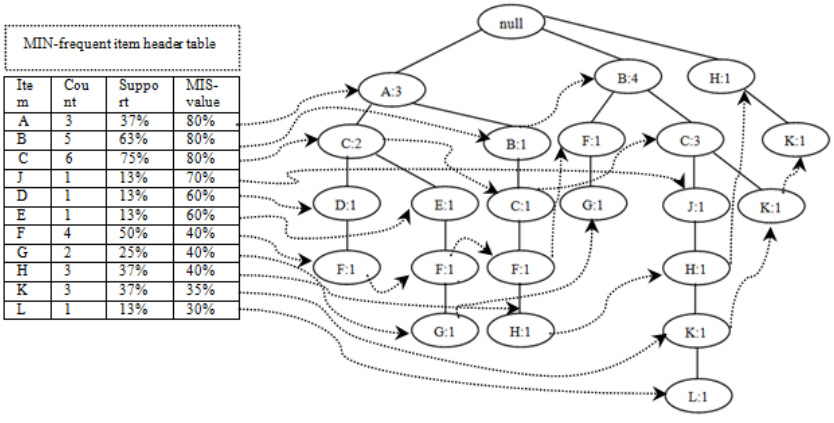


Figure 3.16. Dynamic MIS-tree after adding d with new items.

Now the *Dynamic MIS-tree* in the Figure 3.16 is ready for mining after applying pruning and merging. Mining is done using CFP-Growth++ [3].

3.3.2.4 Adding Increments with Deletions

```
INPUT: MIS-tree, Sorted Minimum items support thresholds MIS_sorted,
incremental database d

OUTPUT: Dynamic MIS-tree

BEGIN

1     Scan d

2     FOR each transaction T in d do:

3       Sort all items in T according to MIS_sorted

4       Delete T from the tree

5     END FOR

6     Calculate the support of all items

7     Update the supports in the Min frequent item header table

8     Return Dynamic MIS-tree

END
```

Figure 3.17. Update process in Dynamic MIS2 for deletions.

Let us explain the pseudo code of the update process for deletions which given in Figure 3.17.When new transactions arrive, they are scanned to be deleted from the tree as follows. First, the items in the new transaction are sorted in decreasing order as the $\mathbf{MIS_{sorted}}$ list. Then transaction is deleted from the tree, such that transactions that has count greater than one are decremented by one, otherwise; nodes are deleted. Update each item's count in this transaction by decrementing its count in the header table by 1. Then link the nodes of same item all through the tree and to the header table. This entire process is indicated in Lines 1-5. The supports of all items in the whole database are calculated then updated in the header tables as in Lines 6-7.

Table 3.13. Incremental database d with deletions.

TID	Item bought	Item bought (ordered)
100	D,C,A,F	A ,C, D, F
400	B,F,G	B, F,G

Table 3.13 shows two transactions to be deleted from the original tree. The right most column of it lists all the items in each transaction following this order according to their MIS values in decreasing order as Table 3.2.

Here we will apply the deletion example on the tree of Figure 3.12.Deletions in Table 3.13 are scanned. And then deleted from the tree as in Figure18, some items' counts are decremented. From these counts supports are calculated and updated in the tables of tree.

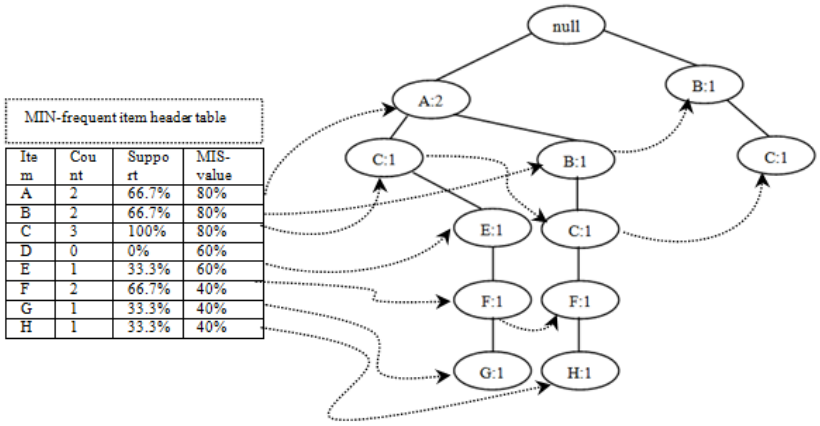


Figure 3.18. The MIS-tree after deletions.

Now the *Dynamic MIS-tree* contains the complete information for all the database transactions and their items' count and support. Once *Dynamic MIS-tree* is built, new transaction can be added to and deleted from the tree and frequent patterns can be mined with multiple support values without rebuilding the tree. Mining steps are the same as CFP-Growth++ [3] as explained in the previous section.

CHAPTER 4

PERFORMANCE EVALUATION

In this chapter, we compare the performance of *Dynamic MIS1, Dynamic MIS2* and CFP-Growth++ algorithms. We are not considering MSApriori and CFP-Growth algorithms for comparison because it has been shown that CFP-Growth++ algorithm is better than the corresponding MSApriori and CFP-Growth algorithms, respectively [3]. However CFP-Growth++ does not have an update feature, it runs from the beginning on the updated database, while our two dynamic algorithms only run on the updates. All algorithms are implemented in C#. All experiments are executed on an Intl(R) core i7 - 5500u CPU@ 2.40 GHz with 8 GB main memory, running on Microsoft Windows 10 operating system.

During performance evaluation, it is ensured that the system state is similar in all test runs and they give similar results when they are repeated. Four datasets are used as in the performance evaluation of this work.

This chapter is divided into seven subsections where in the first, the datasets are presented. In the second subsection, the complexity analysis is done,in the third subsection the static parts of the *Dynamic MIS1* and *Dynamic MIS2* algorithms are compared to CFP-Growth++ algorithm when we vary β (a parameter that controls how the MIS values for items should be related to their frequencies). In the fourth subsection comparison is done on the increments with additions. In the fifth subsection, we measure the performance of the algorithms on the increments of additions with new items. In the sixth subsections, a comparison is done on the increments with deletions. In the last subsections, a discussion on results is done.

4.1 Properties of datasets

Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ algorithms are tested on four datasets in order to measure their performances on datasets having different characteristics. Two real datasets (D1 and D4) and two synthetic datasets (D2 and D3) are used in the experiments. Table 4.1 displays the information about the properties of

the datasets: average size of the transactions (T), number of transactions (D) and number of items (N) and the density[1] of a dataset that indicates the similarity of the transactions. The real dataset (D1) is taken from Frequent Itemset Mining Implementations Repository [FIMI] in [35], the synthetic dataset (D2) is generated using the generator by the IBM Almaden Quest research group, and we generate the last synthetic dataset (D3) using the data generator software in [35]. Real dataset Kosarak (D4) which is a very large dataset containing 990002 sequences of click-stream data from hungarian news portal and a very large number of items [35].

Table 4.1. Properties of datasets.

Dataset	Type	T	D	N	Density%
D1 (Retail)	Real	10.3	88162	16470	0.06
D2 (T40I1D100K)	synthetic	40	100K	942	4.25
D3	synthetic	1.1	100K	5356	0.02
D4 (Kosarak)	Real	8.1	990002	41270	0.02

D1, D2 and D4 are used for testing the performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ algorithms. The previous datasets are used for measuring the performance of increments with additions and deletions. Finally, since increments with additions with new items need to have new items with their MIS values in each increment, we generate D3 for this purpose.

Generating MIS values

For our experiments, we need a method to assign MIS values to items in the data set. We use the actual frequencies (or the supports) of the items in the dataset as the basis for MIS assignments. Specifically, we use the following formulas:

$$MIS(i) = \begin{cases} M(i) & M(i) > LS \\ LS & Otherwise \end{cases}$$

$$M(i) = \beta f(i)$$

f(i) is the actual frequency (or the support expressed in percentage of the data set size) of item i in the data. LS is the user-specified lowest minimum item support allowed. β

[1] Density (%) = (Average Transaction Length / # of Distinct Items) × 100

($0 \leq \beta \leq 1$) is a parameter that controls how the MIS values for items should be related to their frequencies. Thus, to set MIS values for items we use two parameters, β and LS. If $\beta = 0$, we have only one minimum support, LS, which is the same as the traditional association rule mining. If $\beta = 1$ and $f(i) \geq LS$, f(i) is the MIS value for i [17]. This formula is used to generate MIS values to algorithms which use multiple support thresholds as in [2, 3, 17 and 33].

4.2 Complexity analysis of algorithms

Computational complexity of building the initial tree is same for both algorithms. It is (T * V); where T is the number of transactions, and V the average transaction length. It is reasonable to conclude that building the tree is directly proportional to the density of the dataset.

The complexity of the pruning procedure in CFP-Growth++ is O (N * C) where N is the number of nodes holding the items to be pruned, C is the num-ber of their children. However in Dynamic MIS the pruning procedure is replaced by relocating items between header tables which has a complexity of O (N) where N is the number of items to be transferred; it is linear and much less than that of pruning in CFP-Growth++. The merging procedure in CFP-Growth++ is O (N^2 * K) where N is number of nodes in the tree and K is the node links, however this high complexity is skipped in Dynamic MIS since it has no merging procedure.

The complexity of adding increments to the tree (Addition, Addition with new items and Deletion) is O (T * V) where T is the number of the incremental transactions, and V the average transaction length, so it is proportional to the number of transactions in the incremental DB d and its density.

In terms of space complexity; Dynamic MIS needs more space since it keeps the whole tree in memory while CFP-Growth++ keeps a compact pruned tree.

4.3 Performance of the static algorithms

To test the execution time and memory usage performance of the static version of our algorithms (Dynamic MIS1 Builder and Dynamic MIS2 Builder), we only compare Dynamic MIS1 Builder with the CFP-Growth++ algorithm since Dynamic MIS2 Builder algorithm is the same as CFP-Growth++. We use three datasets; real life datasets D1, D4 and synthetic dataset D2. We choose LS = 0.01 and varies ten values of β (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1). We varied β values against the execution time and memory usage of the algorithms. In other words we varied the MIS values of the items in the dataset. From the formula of generating MIS values; we conclude that as β increases the items have higher MIS values as a result the number of frequent patterns decreases.

4.3.1 Execution time

It is clear from Figure 4.1 that when the Retail dataset is used, the execution time performance for the Dynamic MIS1 builder algorithm is better than CFP-Growth++. For different values of β, when β increases; the number of frequent itemsets decreases, as a result the execution time of the two algorithms decreases. Dynamic MIS1 Builder algorithm achieves reasonable success since it does not apply tree pruning and merging which is time consuming, it only applies mining for the items that exist in the primary header table. The execution time values of Dynamic MIS1 Builder seems as they have the same values, but actually these values are different and they decrease from 2.06 to 1.98 (they are very close to each other).

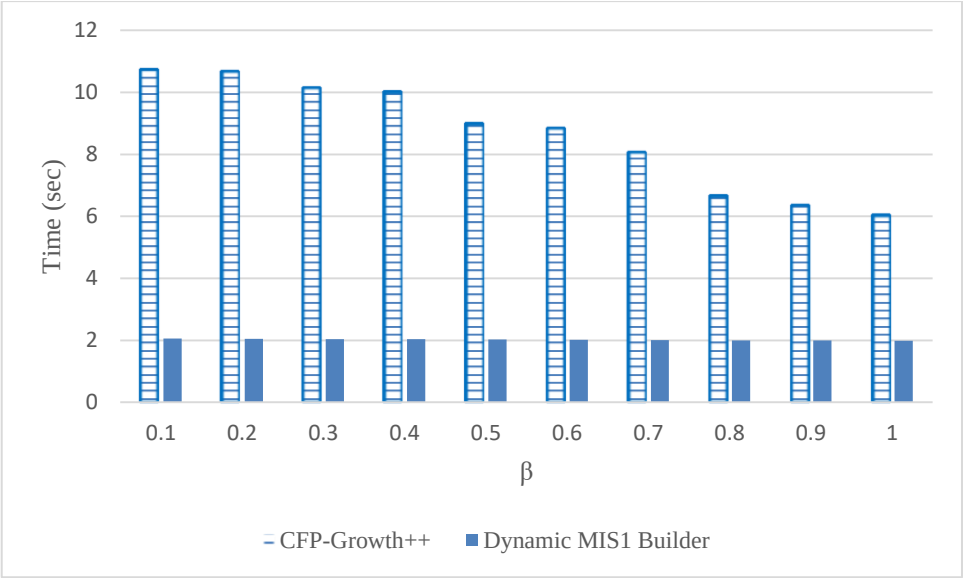


Figure 4.1. Execution time on the dataset D1 (Retail).

The execution time performance of Dynamic MIS1 Builder and CFP-Growth++ on D2 is illustrated in Figure 4.2. As β increases; the number of frequent itemsets decreases, as a result the execution time of the two algorithms decreases. Dynamic MIS1 builder algorithm is slightly better than CFP-Growth++. Since it does not apply tree pruning and merging which is time consuming, it only applies mining for the primary header table items.

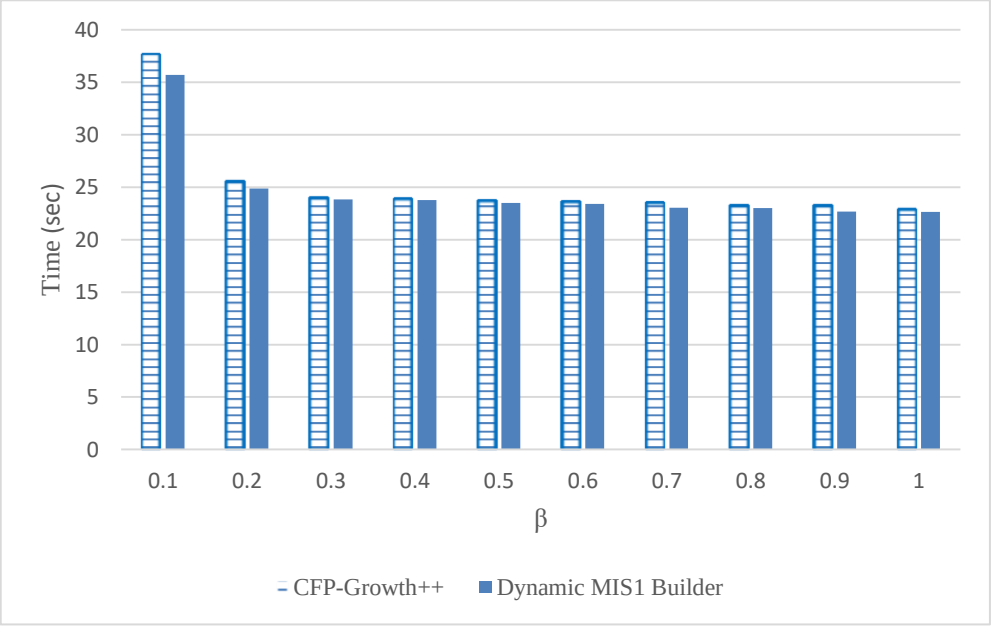


Figure 4.2. Execution time on the dataset D2 (T40i10d100K).

The execution time performance of Dynamic MIS1 Builder and CFP-Growth++ on D4 with different values of β is demonstrated in Figure 4.3. While β increases; the number of frequent itemsets decreases, as a result the execution time of the two algorithms decreases. In every value of β, Dynamic MIS1 Builder performs better than CFP-Growth++. Dynamic MIS1 Builder algorithm achieves reasonable success since it does not apply tree pruning and merging, it only applies mining for the items that located in the primary header table. The execution time values of Dynamic MIS1 Builder are very close to each other and they are decreased from 35.64 to 33.17 (a small decreasing rate) since the process of transferring items between the two header tables takes a small time.

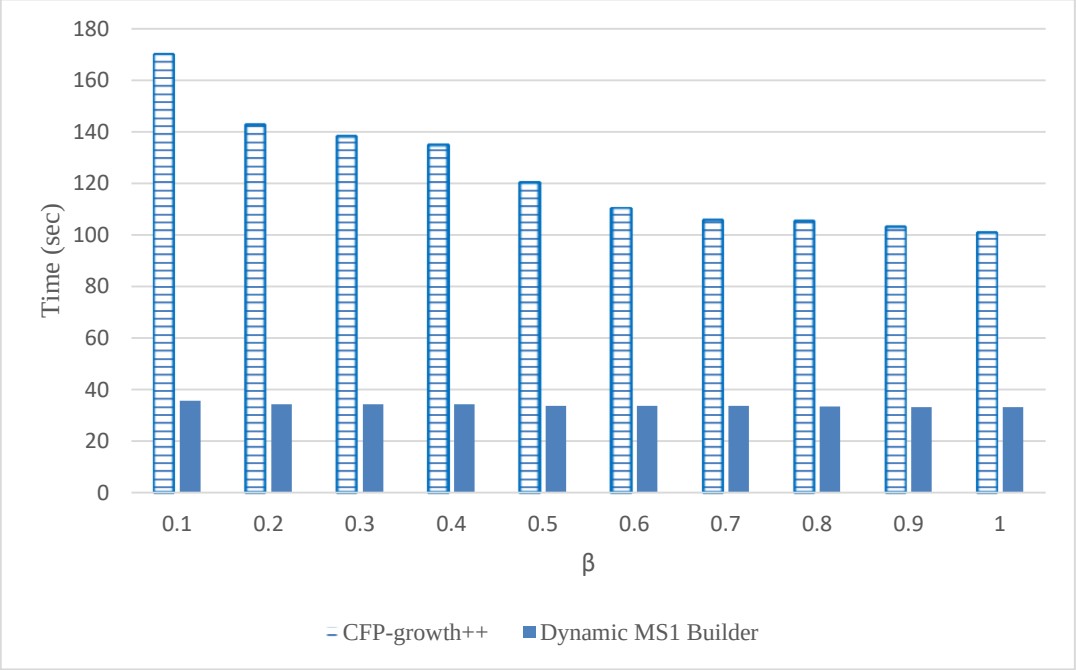


Figure 4.3. Execution time on the dataset D4 (Kosarak).

From Figure 4.1, Figure 4.2 and Figure 4.3; the difference in execution time performance is more clear when the datasets D1 and D4 are used, this is due to the nature of the datasets, such that they are sparse, and have larger number of items with a variant length of transactions.

The speed-up table for static versions is shown in Table 4.2. The speed-up decreases from 5.22 to 3.05 on D1 for MIS1 Builder algorithm, while β values increase from 0.1 to 1. MIS1 Builder algorithm can be up to 5.02 times better than CFP-Growth++ in terms of execution time when it is executed on the real dataset D1. The

reason for this is the nature of this real dataset and structure of the MIS1 Builder algorithm which uses two header tables for the items that enable the mining process of the algorithm to be executed without pruning and merging as CFP-Growth++. When the algorithms are executed on D2, the speed-up of MIS1 Builder algorithm decreases from 1.06 to 1.01 while β values increase, this is due to the nature of synthetic dataset D2. When the algorithms are executed on D4, the speed-up of MIS1 Builder algorithm decreases from 4.78 to 3.05 while β values increase, this is due to the difference in structure of the two algorithms and the nature of the dataset D4.

Table 4.2. Speed-up table for static versions.

Dataset	β	Speed-up with MIS1 Builder[2]
D1 (Retail)	0.1 - 1	5.22 - 3.05
D2 (T40I1D100K)	0.1 - 1	1.06 - 1.01
D4 (Kosarak)	0.1 - 1	4.78 - 3.05

4.3.2 Memory usage

It is illustrated in Figure 4.4 that the memory usage of MIS1 Builder algorithm is a little bit higher than or equal to the memory usage of CFP-Growth++ algorithm, since it keeps the whole tree in memory without pruning and merging. CFP-Growth++ algorithm apply pruning and merging for the tree, therefore it has memory usage less than or equal to MIS1 Builder algorithm.

[2] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of MIS1 Builder algorithm.

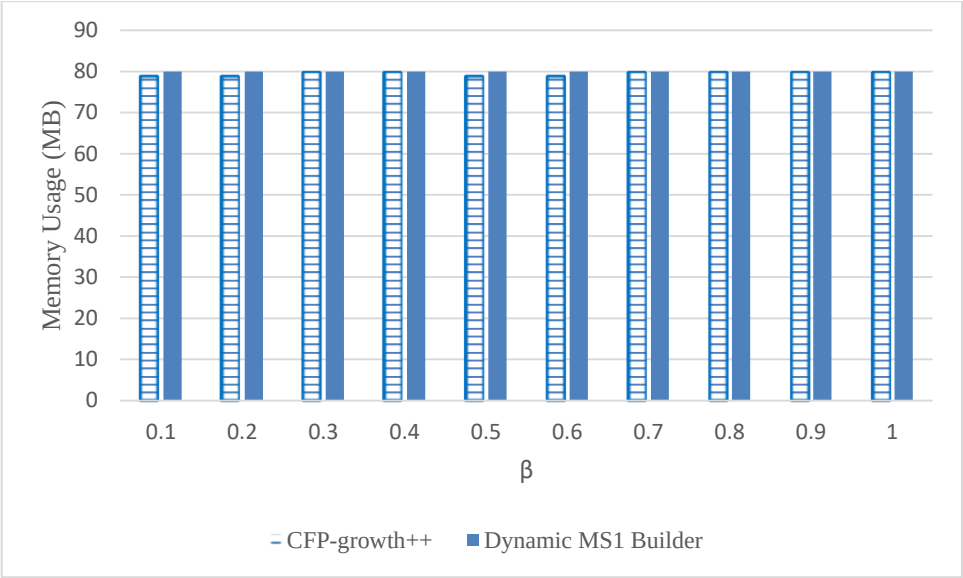


Figure 4.4. Memory usage on dataset D1 (Retail).

Figure 4.5 shows that the Dynamic MIS1 Builder algorithm has the highest memory usage in most cases since it keeps the whole tree in memory without pruning and merging. Since CFP-Growth++ and applies pruning and merging for the tree, it has memory usage less than MIS1 Builder algorithm.

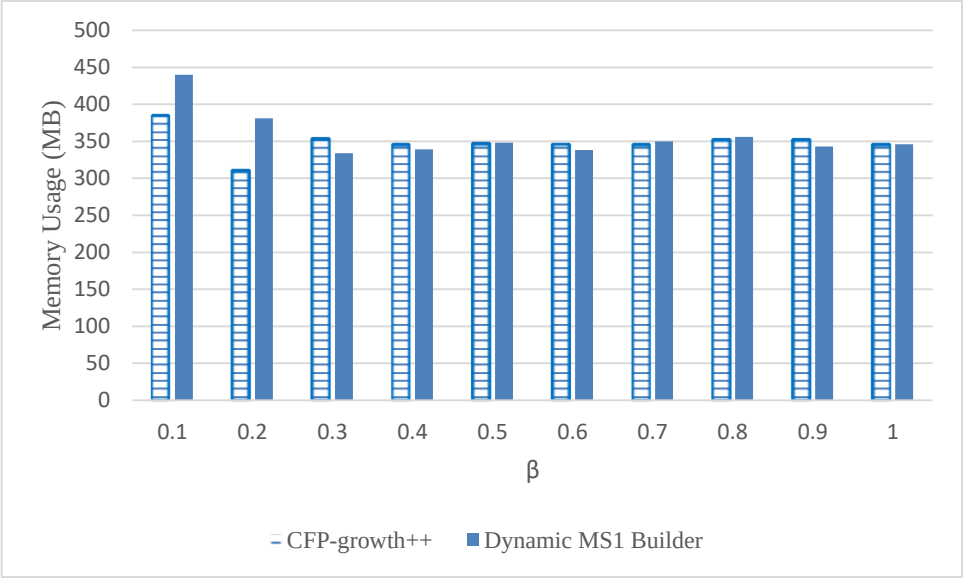


Figure 4.5. Memory usage on synthetic dataset D2 (T40i10d100K).

The memory usage performance of Dynamic MIS1 Builder and CFP-Growth++ on D4 is demonstrated in Figure 4.6. The memory usage of MIS1 Builder algorithm is

a little bit higher than or equal to the memory usage of CFP-Growth++ algorithm, since it keeps the whole tree in memory without pruning and merging. CFP-Growth++ algorithm apply pruning and merging for the tree, therefore it has memory usage less than or equal to MIS1 Builder algorithm.

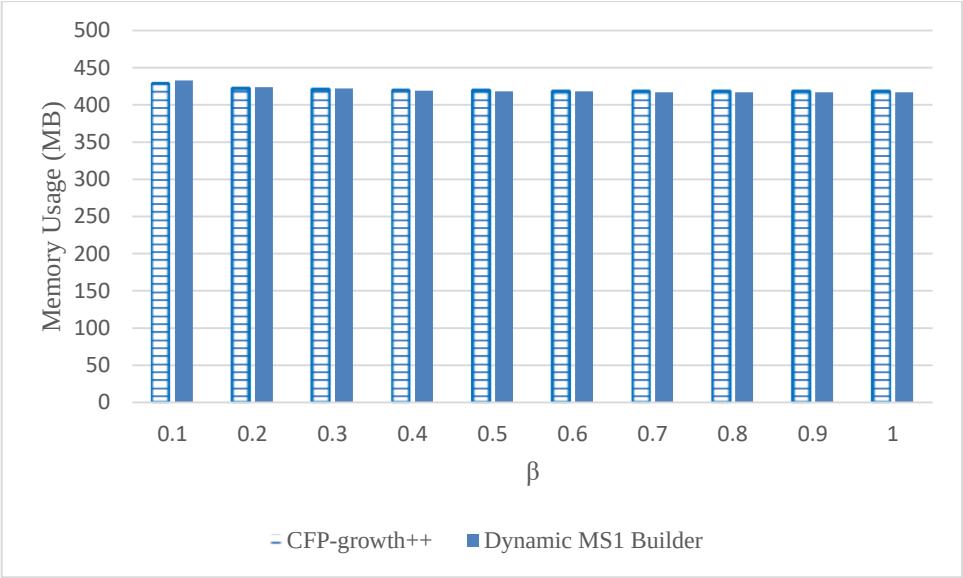


Figure 4.6. Memory usage on dataset D4 (Kosarak).

The memory gain table for static versions is given in Table 4.3. The memory gain increases from (- 1.27) to 0.00 on D1 for MIS1 Builder algorithm while β values increase from 0.1 to 1, so for β = 0.1; MIS1 Builder algorithm is 1.27 times less than CFP-Growth++ in terms of memory usage when it is executed on the real dataset D1, for β = 1; MIS1 Builder algorithm memory usage is equal to it in CFP-Growth++ on D1. The reason for this is due to the nature of the Real dataset and structure of the MIS1 Builder algorithm which uses two header tables for the items that enable the algorithm to execute mining without pruning and merging whereas CFP-Growth++ keeps the whole tree in memory. When the algorithms are executed on D2, the memory of MIS1 Builder algorithm increases from (-14.29) to (-0.29), this is due to the nature of synthetic dataset D2. The memory gain increases from (- 0.93) to 0.24 on D4 for MIS1 Builder algorithm while β values increase from 0.1 to 1, this memory gain values are due to the nature of D4 and the structure of each algorithm.

We conclude that MIS1 Builder algorithm memory usage is higher than or equal to the memory usage of CFP-Growth++, since MIS1 Builder algorithm keeps the whole tree in memory, however CFP-Growth++ keeps minimized pruned tree in memory.

Table 4.3. Memory gain table for static versions.

Dataset	β	Memory-gain with MIS1 Builder[3]
D1 (Retail)	0.1 - 1	(- 1.27) - 0.00
D2 (T40I1D100K)	0.1 - 1	(- 14.29) - (-0.29)
D4 (Kosarak)	0.1 - 1	(- 0.93) - 0.24

4.4 Execution time on increments (additions)

In this experiment we compare the execution time performance of the algorithms on the increments with additions; dynamic algorithms (MIS1 and MIS2) and CFP-Growth++. For this purpose; two real datasets D1 and D4 and one synthetic dataset D3 are used.

In the increments with additions tests, we split each dataset into two parts. The part with D = (100 - x)% from the beginning of the transactions forms the initial dataset and the remaining part with d = x% of the transactions forms the increments. This subsection includes the performance analysis of the algorithms on datasets varying the x (d). The purpose is to observe how the addition size of increments affects the performance of the algorithms for the datasets. The MIS values are kept same as those in the addition tests. The execution time of *Dynamic MIS1*, *Dynamic MIS2* and CFP-Growth++ are measured with thirteen split sizes, namely. 1%, 2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, 10%, 11%, 12%, and 13%, for D1. Ten splits are used as 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45% and 50%, for D2. And eighteen splits are used as 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%, 75%, 80%, 85%, and 90%, for D4. In all splits the MIS values are kept same.

[3] Memory-gain = ((Memory usage of CFP-Growth++ algorithm - Memory usage of MIS1 Builder algorithm) / Memory usage of CFP-Growth++)*100

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different addition sizes on the dataset D1 is demonstrated in Figure 4.7. In every increment size, Dynamic MIS1 and Dynamic MIS2 performs better than re-running CFP-Growth++ from the beginning since they only run for the increments. Execution time of Dynamic MIS1 is less than Dynamic MIS2. The reason for this result is due to the structure of the MIS1 Builder algorithm which uses two header tables for the items and that allows this algorithm to execute mining without pruning and merging operations whereas MIS2 Builder algorithm does mining with pruning and merging operations. We conclude that on D1; Dynamic MIS1 performs much better than both of Dynamic MIS2 and CFP-Growth++ in terms of execution time.

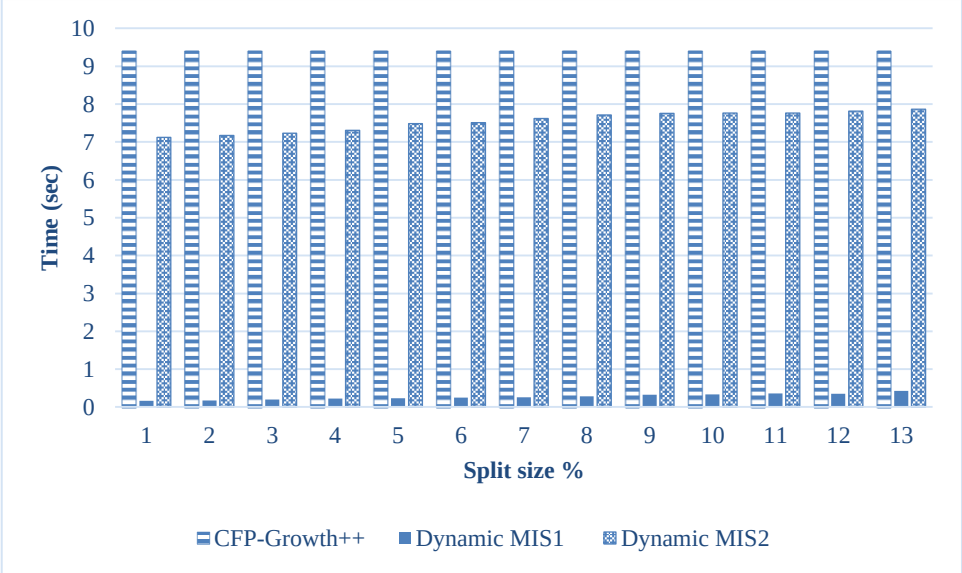


Figure 4.7. Execution time on real dataset D1 (Retail) with increments (additions).

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different addition sizes on Dataset D4 (Kosarak) is illustrated in Figure 4.8. In every increment size, Dynamic MIS1 and Dynamic MIS2 performs better than re-running CFP-Growth++ from the beginning since they are only running for the increment. Dynamic MIS1 execution time is faster than Dynamic MIS2. The reason for this result is the difference between the structure of the two algorithms, such that; MIS1 Builder algorithm uses two header tables for locating the frequent and infrequent items, consequently; the algorithm does mining without pruning and merging operations whereas MIS2 Builder algorithm does mining with pruning and merging operations. Dynamic MIS2 is still better than CFP-Growth++ until the increment size is 85% of the

original size of data, after that size; CFP-Growth++ becomes better than Dynamic MIS2 algorithm.

We conclude that on D4; Dynamic MIS1 performs much better than both of Dynamic MIS2 and CFP-Growth++ in terms of execution time.

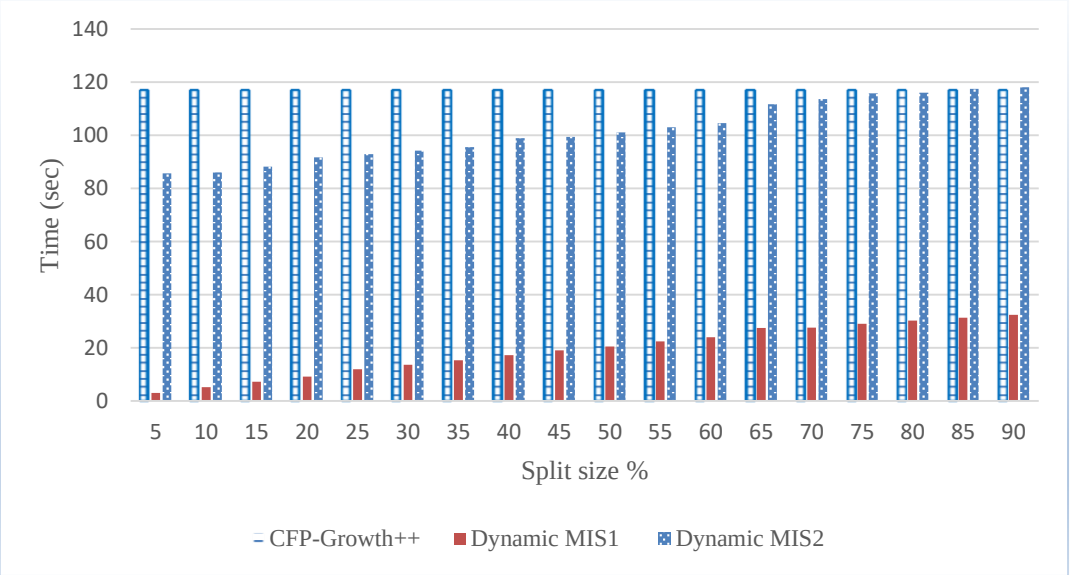


Figure 4.8. Execution time on dataset D4 (Kosarak) with increments (additions).

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different addition sizes on dataset D2 is demonstrated in Figure 4.9. In every increment size, Dynamic MIS1 and Dynamic MIS2 perform better than re-running CFP-Growth++ from the beginning since they only run for the increments. In most cases of splits; Dynamic MIS1 execution time is faster than Dynamic MIS2. The main reason for this result is due to the structure of the MIS1 Builder algorithm, which uses two header tables for the items, thus the algorithm does mining without pruning and merging operations, whereas MIS2 Builder algorithm does mining with pruning and merging operations. We conclude that on D2; Dynamic MIS1 performs much better than both of Dynamic MIS2 and CFP-Growth++ in terms of execution time.

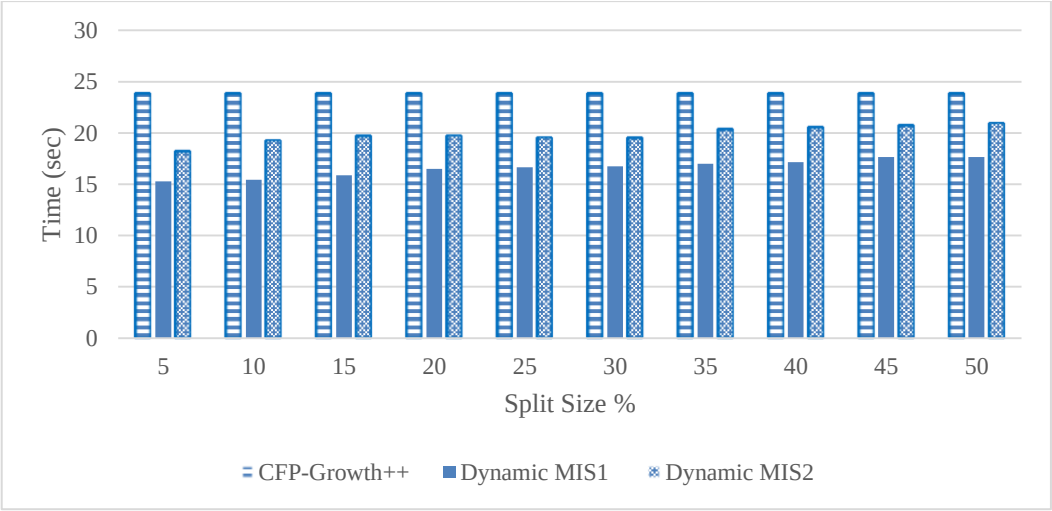


Figure 4.9. Execution time on dataset D2 (T40i10d100K) with increments (additions).

The speed-up by running Dynamic MIS1 and Dynamic MIS2 instead of re-running CFP-Growth++ when the database is updated is displayed in Table 4.4. For Dynamic MIS1; the speed-up increases from 22.21 to 55.94 while the split size decreases on D1. Speed-up of Dynamic MIS1 is from 1.15 to 1.33 on D2, and from 37.67 to 3.61 on D4 respectively. For Dynamic MIS2; the speed-up increases from 1.19 to 1.32 while the split size decreases on D1, and from 1.60 to 1.35 on D2. As the table point outs, the highest speed-up occurs when the Dynamic MIS1 runs on D1, and this range is from 1.37 - 0.99 on D4. The reason for this speed-up over CFP-Growth++ is running Dynamic MIS1 on the addition only instead of running from the beginning. Speed-up of Dynamic MIS1 is higher than Dynamic MIS2 because of the structure of the MIS1 Builder algorithm, which has two header tables for the items, mining is executed without pruning and merging operations, while MIS2 Builder algorithm does mining with pruning and merging operations.

Table 4.4. Speed-up table on increments (addition).

Dataset	Split size %	Speed-up with Dynamic MIS1[4]	Speed-up with Dynamic MIS2[5]
D1 (Retail)	1 – 13	55.94 - 22.21	1.32 - 1.19
D2 (T40I1D100K)	5 – 50	1.56 - 1.35	1.31 - 1.14
D4 (Kosarak)	5 – 90	37.67 - 3.61	1.37 - 0.99

4.5 Execution time on increments (additions with new items)

Here we want to evaluate the execution time performance of increments with additions with new items for the two dynamic algorithms (MIS1 and MIS2) and CFP-Growth++. For this purpose we generate our own dataset D3 using dataset generator (IBM_Quest_data_generator[35]), The properties of D3 are shown in Table 4.1. In this experiment we generate our data set to control the new items that not exist in the original data base and to determine their MIS values, in order to take them as input to our two dynamic algorithms. We use the same strategy mentioned in section 4.4 for splitting the data set. Also we use eighteen split sizes, namely 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%, 75%, 80%, 85% and 90%. And in each run; the incremental part has new items with their new MIS values. In all parts LS (the user-specified lowest minimum item support) and β (parameter that controls how the MIS values for items should be related to their frequencies) are still the same values. To allow variation in MIS values we choose (beta = 0.5 and LS = 0.01). The number of new items in each split is constant and equal to 100. We measure the size of d (split) against the time for each algorithm.

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different addition sizes with new items for each addition on Dataset D3 is demonstrated in Figure 4.10. In every increment size, Dynamic MIS1 and Dynamic MIS2 perform better than re-running CFP-Growth++ from the beginning since they are only running for the increment. In most cases of splits; Dynamic MIS1 is faster than Dynamic MIS2. The reason for this result is due to the structure of the MIS1 Builder algorithm which uses two header tables for the items that enable the algorithm to

[4] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS1 algorithm.
[5] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS2 algorithm.

execute mining without need to pruning and merging operations as MIS2 Builder algorithm. Dynamic MIS2 is faster than CFP-Growth++ for all splits before the split size 90%. We conclude that on D3; Dynamic MIS1 has the best performance in terms of execution time.

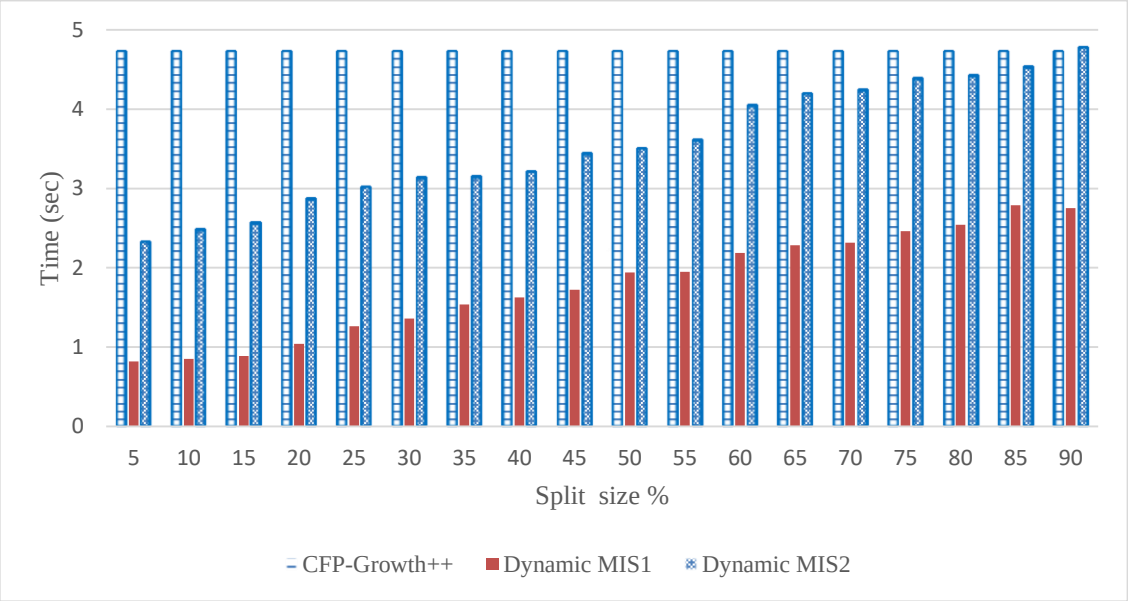


Figure 4.10. Execution time on dataset (D3) with increments (additions with new items).

The speed-up by running Dynamic MIS1 and Dynamic MIS2 instead of re-running CFP-Growth++ when the database is updated is shown in Table 4.5. For Dynamic MIS1; the speed-up decreases from 5.76 to 1.72 while the split size increases in D3. For Dynamic MIS2; the speed-up increases from 0.99 to 2.03 while the split size decreases in D3. As the table point outs, the Dynamic MIS1 and Dynamic MIS2 have higher speed-up than CFP-Growth++, the reason for this speed-up over CFP-Growth++ is dynamic aspect of algorithms; Dynamic MIS1is running on the addition only instead of running from beginning. Also the speed-up of Dynamic MIS1 is slightly higher than in Dynamic MIS2 because of the structure of the MIS1 Builder algorithm which has two header tables for the items, so the algorithm executes mining without pruning and merging operations as MIS2 Builder algorithms.

Table 4.5. Speed-up table on increments (addition with new items).

Dataset	Split size %	Speed-up with Dynamic MIS1[6]	Speed-up with Dynamic MIS2[7]
D3 (synthetic)	1 – 13	5.76 - 1.72	2.03 - 0.99

4.6 Execution time on increments (deletions)

The last comparison is to determine how the size of deletions affects the performances of algorithms. In these tests, we use the real datasets D1, D4 and the synthetic dataset D2. First we split the dataset into two parts like the addition tests. In this experiment we will run the three algorithm. The part with D = 100% from the beginning of the transactions forms the initial dataset and the remaining part with d = x% of the transactions forms the additions. The x is set to 20 for the dataset D1 during the tests. In other words, for the tests on D = 100% of the transactions of D1 are the initial dataset and 20% of the transactions of D1 are the additions with deletions. In case of running CFP-Growth++; the number of transactions of dataset for will equal (D - d)% from beginning. The MIS values are kept same as those in the addition tests.

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different deletion sizes on Dataset D1 is demonstrated in Figure 4.11. Dynamic MIS2 performs better than CFP-Growth++ before split size 10% while Dynamic MIS1 has better execution time than both CFP-Growth++ and Dynamic MIS2 in all splits in our experiment. In every increment size, Dynamic MIS1 performs better than re-running CFP-Growth++ from the beginning since they it only running for the increment. In most cases of splits; Dynamic MIS1 execution time faster than Dynamic MIS2. The reason for this result is due to the structure of the MIS1 Builder algorithm which uses two header tables for the items that enable the algorithm to execute mining without pruning and merging operations as MIS2 Builder algorithms. We conclude that on D1; Dynamic MIS1 has the best performance while CFP-Growth++ has better performance than Dynamic MIS2 starting from 10% of the incremental size.

[6] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS1 algorithm.
[7] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS2 algorithm.

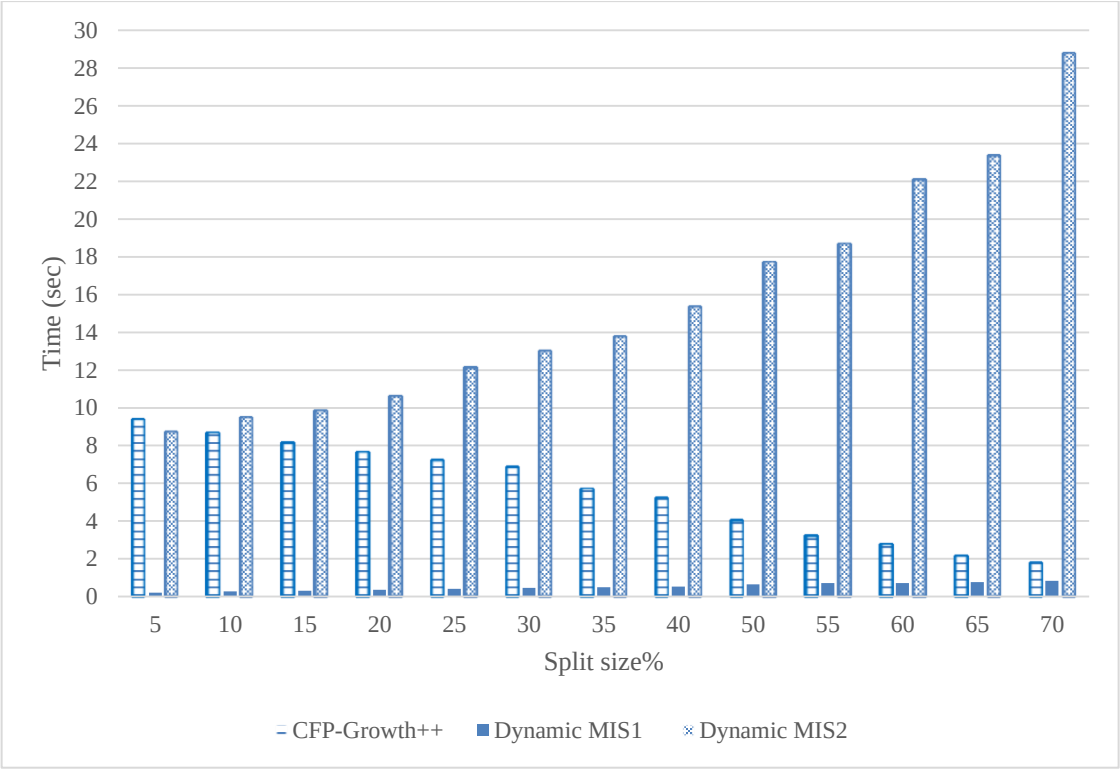


Figure 4.11. Execution time on dataset D1 (Retail) with increments (deletions).

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different deletion sizes on Dataset D4 is demonstrated in Figure 4.12. Dynamic MIS2 performs better than CFP-Growth++ before split size 15% while Dynamic MIS1 has better execution time than both CFP-Growth++ and Dynamic MIS2 in all splits in our experiment. In every increment size, Dynamic MIS1 performs better than re-running CFP-Growth++ from the beginning since they it only running for the increment. In most cases of splits; Dynamic MIS1 execution time faster than Dynamic MIS2. The reason for this result is due to the structure of the MIS1 Builder algorithm which uses two header tables for the items that enable the algorithm to execute mining without pruning and merging operations as MIS2 Builder algorithms. We conclude that on D4; Dynamic MIS1 has the best performance while CFP-Growth++ has better performance than Dynamic MIS2 starting from 15% of the incremental size.

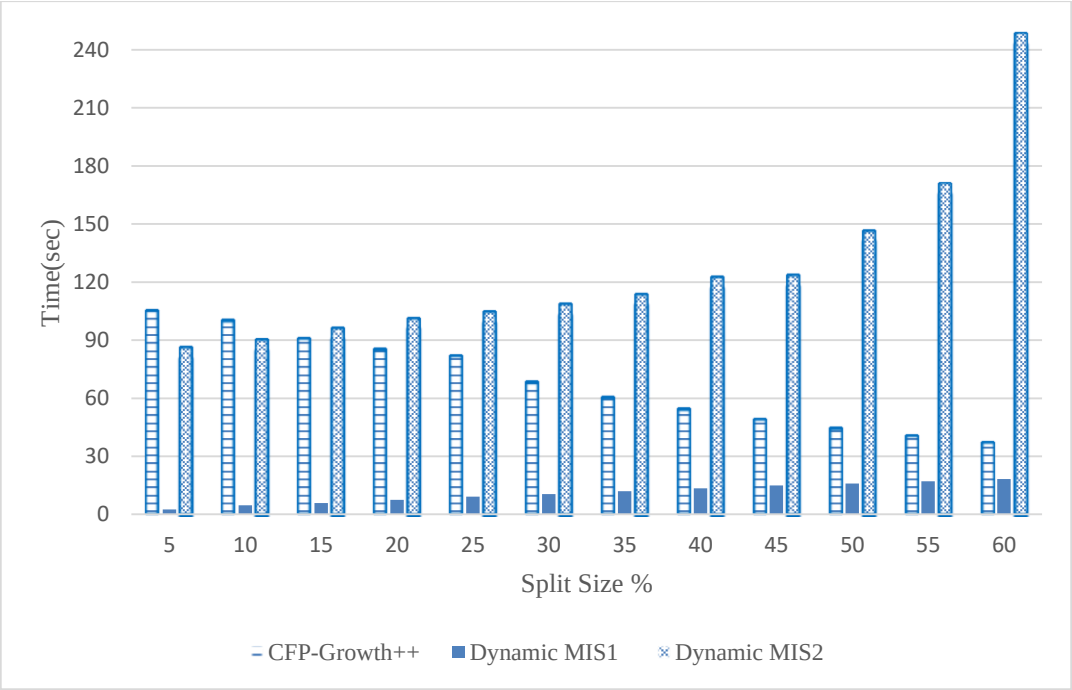


Figure 4.12. Execution time on dataset D4 (Kosarak) with increments (deletions).

The execution time performance of Dynamic MIS1, Dynamic MIS2 and CFP-Growth++ with different deletion sizes on Dataset D2 is illustrated in Figure 4.13. CFP-Growth++ has higher execution time than both dynamic MIS1 and Dynamic MIS2 in the splits until 20%. From 25% to 50% CFP-Growth++ has less execution time than both dynamic MIS1 and Dynamic MIS2. Starting from 55% to the last split CFP-Growth++ execution time become higher than the other two dynamic since the number of frequent items equal 0. In all splits dynamic MIS1 is a slightly better than dynamic MIS2. This result is due to the structure of the algorithms and the properties of the dataset D2 which are dense and has less number of items, and it is observed from the results that on each increment size, the frequent itemsets count decreases in a marked rate, which can be up to half of it in the previous increment in some cases. Thus; in our dynamic algorithms, the time spent in mining process becomes less while the time for deleting these increments increases, but this rate of time increasing is much less than the rate of time decreasing in mining, so the total execution time for the our two dynamic algorithms decreases while the increment splits increases

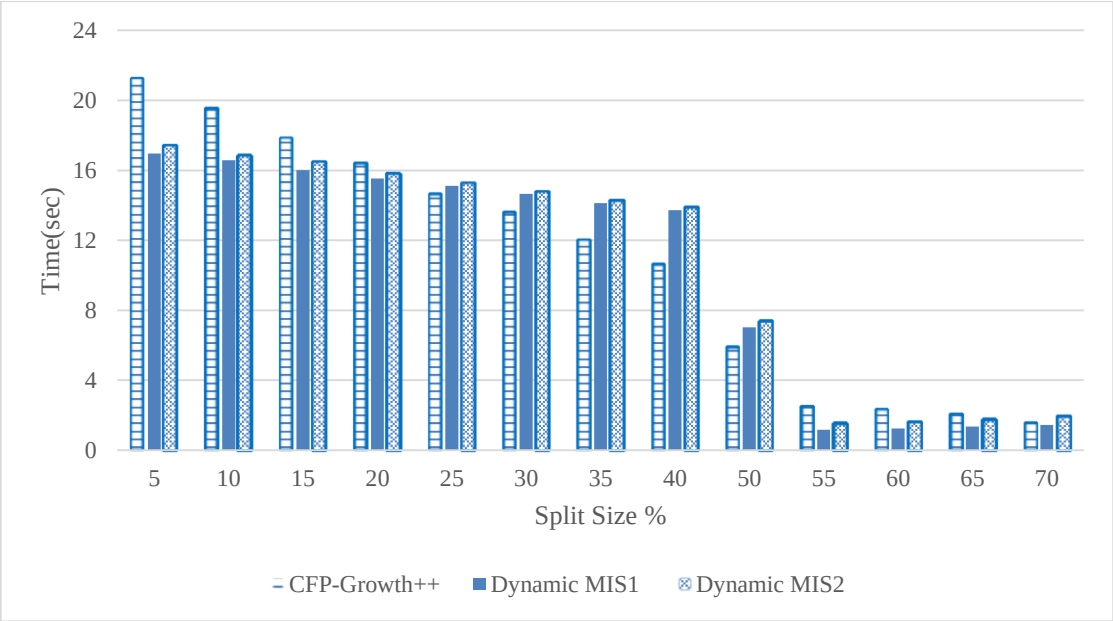


Figure 4.13. Execution time on dataset D2 (T40i10d100K) with increments (deletions).

The speed-up by running Dynamic MIS1 and Dynamic MIS2 instead of re-running CFP-Growth++ when the database is updated for deletion is shown in Table 4.6. For Dynamic MIS1; the speed-up increases from 2.26 to 44.88 while the split size decreases in D1. For Dynamic MIS2; the speed-up increases from 1.07 to 1.29 while the split size decreases in D1. For Dynamic MIS1; the speed-up increases from 2.06 to 40.16 while the split size decreases in D4. For Dynamic MIS2; the speed-up increases from 0.15 to 1.28 while the split size decreases in D4. For Dynamic MIS1; the speed-up increases from 1.12 to 1.25 while the split size decreases in D2. For Dynamic MIS2; the speed-up increases from 0.84 to 1.22 while the split size decreases in D2. As the table point outs, the Dynamic MIS1 and Dynamic MIS2 have higher speed-up than CFP-Growth++, the reason for this speed over CFP-Growth++ is Dynamic MIS1, Dynamic MIS1are running on the addition only instead of running from beginning. Also the speed-up of Dynamic MIS1 is higher than it in Dynamic MIS2 because of the structure of the MIS1 Builder algorithm which has two header tables for the items that allow the algorithm to execute mining without pruning and merging operations. The minimum speed-up of Dynamic MIS2 algorithm less than one since CFP-Growth++ has better performance than Dynamic MIS2 after 10% on D1 and after 15% on D4 of the incremental size. Speed-up with Dynamic MIS1 on D2 is very small compared with it onD1 and D4, the reason for this is due to the nature of the synthetic dataset D2 which

are dense and has less number of items, whereas D1 and D4 are sparse and have larger number of items.

Table 4.6. Speed-up table on increments (deletions).

Dataset	Split size %	Speed-up with Dynamic MIS1[8]	Speed-up with Dynamic MIS2[9]
D1 (Retail)	5 – 70	44.88 - 2.26	1.07 - 0.07
D4 (Kosarak)	5 – 60	40.16 - 2.06	1.28 - 0.15
D2 (T40I1D100K)	5 – 70	1.25 - 1.12	1.22 - 0.84

4.7 Discussion on results

In this research, we compare the performance of *Dynamic MIS1*, *Dynamic MIS2* and CFP-Growth++ algorithms. We first compare the execution time and memory usage performance of (*Dynamic MIS1* and CFP-Growth++) algorithms on static databases, then we compare the execution time performance of (*Dynamic MIS1*, *Dynamic MIS2* and CFP-Growth++) algorithms on dynamic databases. We generate a synthetic dataset D3 to be used in the experiment of increments with additions with new items. In all other experiments; we use three datasets; real life datasets D1, D4 and synthetic dataset D2.

In the first experiment, the execution time performance of the static version of our algorithm Dynamic MIS1 Builder is compared with the CFP-Growth++ algorithm. We choose LS = 0.01 and varies ten values of β. As β increases the items have higher MIS values as a result the number of frequent patterns decreases. The speed-up decreases from 5.22 to 3.05 on D1 for MIS1 Builder algorithm, while β values increase from 0.1 to 1. MIS1 Builder algorithm can be up to 5.02 times better than CFP-Growth++ in terms of execution time when it is executed on the dataset D1. When the algorithms are executed on D2, the speed-up of MIS1 Builder algorithm decreases from 1.06 to 1.01. When the algorithms are executed on D4, the speed-up of MIS1 Builder algorithm decreases from 4.78 to 3.05 while β values increase, these results are due to the nature

[8] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS1 algorithm.
[9] Speed-up = Execution time of CFP-Growth++ algorithm / Execution time of Dynamic MIS2 algorithm.

of each dataset ,the difference in structure of the algorithms and the variation of number of frequent patterns.

In the second experiment, the memory usage performance of the static version of our algorithm Dynamic MIS1 Builder is compared with the CFP-Growth++ algorithm. Ten values of β are varied against the execution time the algorithms. The memory gain increases from (- 1.27) to 0.00 on D1 for MIS1 Builder algorithm while β values increase from 0.1 to 1, so for β = 0.1; MIS1 Builder algorithm consumes 1.27 times more memory than than CFP-Growth++ when it is executed on the real dataset D1, for β = 1; MIS1 Builder algorithm memory usage is equal to it in CFP-Growth++ on D1. When the algorithms are executed on D2, the memory of MIS1 Builder algorithm increases from (-14.29) to (-0.29). On D4; the memory gain increases from (- 0.93) to 0.24 for MIS1 Builder algorithm while β values increase from 0.1 to 1. We conclude that MIS1 Builder algorithm memory usage is higher than or equal to the memory usage of CFP-Growth++, since MIS1 Builder algorithm keeps the whole tree in memory, however CFP-Growth++ keeps minimized pruned tree in memory.

In the third experiment, we compare the execution time performance of the algorithms on the increments with additions on the dynamic algorithms (MIS1 and MIS2) and base algorithm CFP-Growth++. In the increments with additions tests, we split each dataset into two parts. The part with D = (100 - x)% from the beginning of the transactions forms the initial dataset and the incremental part with d = x% of the transactions forms the increments. The purpose is to observe how the addition size of increments affects the performance of the algorithms for the datasets. The execution time of *Dynamic MIS1, Dynamic MIS2* and CFP-Growth++ are measured with thirteen split sizes. For Dynamic MIS1; the speed-up increases from 22.21 to 55.94 while the split size decreases on D1. Speed-up of Dynamic MIS1 is from 1.15 to 1.33 on D2, and from 37.67 to 3.61 on D4 respectively. For Dynamic MIS2; the speed-up increases from 1.19 to 1.32 while the split size decreases on D1, and from 1.60 to 1.35 on D2. As the values point outs, the highest speed-up occurs when the Dynamic MIS1 runs on D1, and this range is from 1.37 - 0.99 on D4. The reason for this speed-up over CFP-Growth++ is running Dynamic MIS1 on the addition only instead of running from the beginning. Speed-up of Dynamic MIS1 is higher than Dynamic MIS2 because of the structure of the MIS1 Builder algorithm, which has two header tables for the items, mining is executed without pruning and merging operations, while MIS2 Builder algorithm does mining with pruning and merging operations.

In the fourth experiment, we compare the execution time performance of increments with additions with new items for the two dynamic algorithms (MIS1 and MIS2) and CFP-Growth++. We use the same strategy mentioned in the third experiment for splitting the data set. Also we use eighteen split sizes (5% to 90%). And in each run; the incremental part has new items with their new MIS values. In all parts LS and β have the same values (β = 0.5 and LS = 0.01). The number of new items in each split is constant and equal to 100. We measure the size of split against the time for each algorithm. The speed-up decreases from 5.76 to 1.72 while the split size increases in D3. For Dynamic MIS2; the speed-up increases from 0.99 to 2.03 while the split size decreases in D3. As mentioned, the Dynamic MIS1 and Dynamic MIS2 have higher speed-up than CFP-Growth++, the reason for this speed-up over CFP-Growth++ is dynamic aspect of algorithms; Dynamic MIS1 is running on the addition only instead of running from beginning. Also the speed-up of Dynamic MIS1 is slightly higher than in Dynamic MIS2 because of the structure of the MIS1 Builder algorithm which has two header tables for the items, so the algorithm executes mining without pruning and merging operations as MIS2 Builder algorithms.

In the fifth experiment, the comparison is to determine how the size of deletions affects the performances of algorithms. The two dynamic algorithms (MIS1 and MIS2) and CFP-Growth++ run on the datasets D1, D2 and D4. First we split the dataset into two parts. The part with D = 100% from the beginning of the transactions forms the initial dataset and the remaining part with d = x% of the transactions forms the deletion. In case of running CFP-Growth++; the number of transactions of dataset for will equal (D - d)% from beginning. For Dynamic MIS1; the speed-up increases from 2.26 to 44.88 while the split size decreases in D1. For Dynamic MIS2; the speed-up increases from 1.07 to 1.29 while the split size decreases in D1. For Dynamic MIS1; the speed-up increases from 2.06 to 40.16 while the split size decreases in D4. For Dynamic MIS2; the speed-up increases from 0.15 to 1.28 while the split size decreases in D4. For Dynamic MIS1; the speed-up increases from 1.12 to 1.25 while the split size decreases in D2. For Dynamic MIS2; the speed-up increases from 0.84 to 1.22 while the split size decreases in D2. It is observed that, the Dynamic MIS1 and Dynamic MIS2 have higher speed-up than CFP-Growth++, the reason for this speed over CFP-Growth++ is Dynamic MIS1, Dynamic MIS1are running on the increments only instead of running from beginning. Also the speed-up of Dynamic MIS1 is higher than it in Dynamic MIS2 because of the structure of the MIS1 Builder algorithm which has two header tables for

the items that allow the algorithm to execute mining without pruning and merging operations. The minimum speed-up of Dynamic MIS2 algorithm less than one since CFP-Growth++ has better performance than Dynamic MIS2 after 10% on D1 and after 15% on D4 of the incremental size. Speed-up with Dynamic MIS1 on D2 is very small compared with it on D1 and D4, the reason for this is due to the nature of the synthetic dataset D2 which are dense and has less number of items, whereas D1 and D4 are sparse and have larger number of items.

CHAPTER 5

CONCLUSION

The frequent itemset mining algorithms discover the frequent itemsets from a database. But they face many challenges such as accessing data multiple times, response time, huge sizes of database, single support threshold, dynamicity nature of the databases, etc.. Single support threshold that does not allow user to specify support threshold according to the nature of the items. When the database is updated, the frequent itemsets should be updated as well. However, running the frequent itemset mining algorithms with every update is not feasible. Several dynamic update algorithms are proposed but they are devised for single support threshold.

In this study, we focus on dynamic update problem of frequent itemsets under multiple support thresholds; the challenge is to mine the frequent itemsets under multiple support thresholds. In this study, two new dynamic itemset mining under multiple support thresholds algorithms which are called (*Dynamic MIS1* and *Dynamic MIS2*) are introduced and explained, which are tree based, scan the databases only once and avoid the candidate generation problem. They handle increments of additions, additions with new items and deletions. Proposed algorithms are compared with CFP-Growth++ algorithm which is a popular algorithm that is able to find frequent itemsets under multiple support thresholds.

We compare the performance of *Dynamic MIS1*, *Dynamic MIS2* and CFP-Growth++ algorithms. Memory usage performance of our static algorithms is compared, execution time performance of our static and dynamic algorithms are compared. For this purpose; four datasets are used in the experiments. Our findings reveal that in static databases 1) *Dynamic MIS1* achieves up to 5 times speed-up against CFP-Growth++ in terms of execution time since it does not require tree pruning and merging, it only applies mining for items in the primary header table, 2) execution time performance of *Dynamic MIS2* and CFP-Growth++ are similar, 3) memory usage of *Dynamic MIS1* is higher than or equal to CFP-Growth++, since it keeps whole tree in memory,however the other two keep minimized pruned tree in memory. In incremental database 1) *Dynamic MIS1* and *Dynamic MIS2* perform better than rerunning CFP-Growth++ from the begining since they run only on increments, 2) *Dynamic MIS1* is faster than

Dynamic MIS2 and with large sparse database; its speed-up can be up to 55.94, whereas the speed-up of *Dynamic MIS2* cannot exceed 2.03, 3) on large sparse dataset; *Dynamic MIS2* is slightly better than CFP-Growth++ until increment size is 85% of the original size of data, while in small dense dataset until 25% of the original size of data. We conclude that *Dynamic MIS1* algorithm is more efficient than both of *Dynamic MIS2* and CFP-Growth++ and its speed-up is more clear with large sparse datasets.

As a part of our future work, we are planning to enchance our proposed algorithms, and carry on more experiments on different types of datasets, with changing values of MIS values according to different LS and β values, in order to investigate how the proposed algorithms can behave with these changes.

REFERENCES

[1] Adnan, M., Alhajj, R., Barker, K. 2008. "Alternative method for incrementally constructing the FP-Tree", In: Intelligent Techniques and Tools for Novel System Architectures, Studies in Computational Intelligence, Berlin, 109, pp. 361 – 377.

[2] Ya-Han Hu, Y., Chen, Y. 2006. "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism", Decision Support Systems, pp. 42, 1 – 24.

[3] Kiran, R., Reddy, P. 2011. "Novel Techniques to Reduce Search Space in Multiple Minimum Supports-Based Frequent Pattern Mining Algorithms", In: The 14th International Conference on Extending Database Technology, pp. 11 – 20.

[4] Pradeepini, G., Jyothi, S. 2010. "Tree-based incremental association rule mining without candidate itemset generation", Proceedings of the Conference on Trends in Information Sciences & Computing, pp. 78 – 81.

[5] Lin, C., Hong, T., Lu, W., Chien, B. 2008. "Incremental mining with prelarge trees", In: The 21st International Conference on Industrial, EngineeringandOther Applications of AppliedIntelligentSystems. Berlin, Germany, pp. 169 – 178.

[6] Cheungt, D. W., Han, J., Ng, V. T., Wong, C. Y. 1996. "Maintenance of discovered association rules in large databases, An incremental updating technique", In: The 12th International Conference on Data Engineering, pp. 106 – 114.

[7] Cheungt, D. W., Lee, S. D., Kao, B. 1997. "A general incremental technique form maintaining discovered association rules", In: The 5th International Conference on Database Systemsfor Advanced Applications, pp. 185 – 194.

[8] Chandraker, T. , Sao, N. 2012. "Incremental Mining on Association Rules", In: International Journal of Engineering and Science, pp. 31– 33.

[9] Li, T., Li, X. 2010. "IULFP: An efficient incremental updating algorithm based on LFP-tree for mining association rules", In: International Conference on Computer Application and System Modeling, pp. 426 – 430.

[10] Oğuz, D., Ergenç, B. 2012. "Incremental Itemset Mining Based on Matrix Apriori", DEXA-DaWaK, Vienna, Austria, pp.192 – 204.

[11] Oğuz, D., Yıldız B., Ergenç, B. 2013. "Matrix-Based Dynamic Itemset Mining Algorithm", In International Journal of Data Warehousing and Mining (IJDWM), pp.62– 75.

[12] Agrawal, R., Imielinski, T., Swami, A. 1993. "Mining association rules between sets of items in large databases", In: ACM SIGMOD International Conference on Management of Data, pp.207– 216.

[13] Agrawal, R., Srikant, R. 1994. "Fast algorithms for mining association rules in large databases", In: The 20th International Conference on VeryLarge Data Bases. San Francisco, pp. 487 – 499.

[14] Han, J., Pei, J., Yin, Y. 2000. "Mining frequent patterns without candidate generation", In: ACM SIGMOD International Conference on Management of Data, ACM Press, pp.1 – 12.

[15] Pavón, J., Paulo, S., Viana, S. 2006. "Matrix Apriori: Speeding up the search for frequent patterns", In: The 24th IASTED International Conference on Database and Applications. Anaheim, CA: ACTA Press, pp. 75 – 82.

[16] Yıldız, B., Ergenç, B. 2010. "Comparison of Two Association Rule Mining Algorithms without Candidate Generation", In the 10th IASTED International Conference on Artificial Intelligence and Appliations (AIA 2010), Innsbruck, Austia, pp.450 – 457.

[17] Liu, B., Hsu, W., Ma, Y. 1999. " Mining Association Rules with Multiple Minimum Supports", In: The fifth ACM SIGKDD international conference on Knowledge discoveryand data mining, pp. 337 – 341.

[18] Mannila, H. 1998. "Database methods for data mining", KDD-98 tutorial. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and datamining ACM.

[19] Chen, M.S., Han, J. 1996. "Data mining: An overview from a database perspective. IEEE Transaction on knowledge and Data Engineering", pp. 866 – 883.

[20] Mannila, H., Toivonen, H., Verkamo, A.I. 1994. "Efficient algorithms for discovering association rules", In Proc.AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94), Seattle, WA, pp. 181 – 192.

[21] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I. 1996. "Fast discovery of association rules. In Advances in Knowledge Discovery and Data

Mining", U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), AAAI/MIT Press, pp. 307 – 328.

[22] Savasere, A., Omiecinski, E., Navathe, S. 1995. "An efficient algorithm for mining association rules in large databases", In Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95), Zurich, Switzerland, pp. 432 – 443.

[23] Park, J.S., Chen, M.S., and Yu, P.S. 1995. "An effective hash-based algorithm for mining association rules", In Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'95), San Jose, CA, pp. 175 – 186.

[24] Lent, B., Swami, A., and Widom, J. 1997. "Clustering association rules", In Proc. 1997 Int. Conf. Data Engineering (ICDE'97), Birmingham, England, pp. 220 – 231.

[25] Sarawagi, S., Thomas, S., Agrawal, R. 1998. "Integrating association rule mining with relational database systems: Alternatives and implications", In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA, pp. 343 – 354.

[26] Srikant, R., Vu, Q., and Agrawal, R., 1997. "Mining association rules with item constraints", In Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, pp. 67 – 73.

[27] Lakshmanan R., L.V.S., Han, J., Pang, A. 1998. "Exploratory mining and pruning optimizations of constrained associations rules", In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA, pp. 13 – 24.

[28] Grahne, G., Lakshmanan, L., Wang, X. 2000. "Efficient mining of constrained correlated sets", In Proc. 2000 Int. Conf. Data Engineering (ICDE'00), San Diego, CA, pp. 512 – 521.

[29] Feldman, R., Aumann, Y. Lipshtat, O. 1999. "Borders: An Efficient Algorithm for Association Generation in Dynamic Databases", Journal of Intelligent Information System, pp. 61 – 73.

[30] Shan, S., Wang, X., Sui, M. 2010. "Mining Association Rules: A Continuous Incremental Updating Technique", In: International Conference on Web Information Systems and Mining, IEEE Computer Society, pp. 62 – 66.

[31] Dai, B.R., Lin, P.Y. iTM. 2009. "An Efficient Algorithm for Frequent Pattern Mining in the Incremental Database without Rescanning", In: Chien, B.-C., Hong, T.-P., Chen, S.-M., Ali, M. (eds.) IEA/AIE 2009. LNCS. Springer, Heidelberg, pp. 757 – 766.

[32] Cheung, W., Zaiane, R. 2003. "Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint", In: Proc. of the Seventh International Database Engineering and Applications Symposium. IEEE Computer Society

[33] Hoque, F.A., Debnath, M., Easmin, N., Rashad. K. 2011. "Frequent Pattern Mining for Multiple Minimum supports with support Tuning and Tree Maintenance on Incremental Database", Research Journal of Information Technology, pp. 79 – 90.

[34] Han, J., Kamber, M. 2006. "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, pp.157 – 218.

[35] Frequent Itemset Mining Implementations Repository **http://fimi.ua.ac.be/data/**

[36] DARRAB, S., Ergenç, B. 2016. "Frequent Pattern Mining under Multiple Support Thresholds", Wseas Transactıons on Computer Research, pp. 1 – 10.