DEVELOPMENT OF A FRAMEWORK FOR FREQUENT ITEMSET MINING UNDER MULTIPLE SUPPORT THRESHOLDS

A Thesis submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
MASTER OF SCIENCE
in Computer Engineering

by
Sadeq Hussein Saleh DARRAB

July 2016
İZMİR

We approve the thesis of **Sadeq Hussein Saleh DARRAB**

Examining Committee Members:

_____

Assist. Prof. Dr. Belgin Ergenç BOSTANOĞLU
Department of Computer Engineering, Izmir Institute of Technology

_____

Doç. Dr. Adil ALPKOÇAK
Department of Computer Engineering, Dokuz Eylul University

_____

Assist. Prof. Dr. Serap ŞAHİN
Department of Computer Engineering, Izmir Institute of Technology

13 July 2016

_____

Assist. Prof. Dr. Belgin Ergenç BOSTANOĞLU
Supervisor, Department of Computer Engineering
Izmir Institute of Technology

_____

Doç. 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 take this opportunity to express my gratitude to my supervisor Assist. Prof. Dr. Belgin Ergenç Bostanoğlu for her guidance throughout the time of preparation of my thesis. I sincerely thank her for her incessant encouragement as well as for providing essential information and support. I extremely appreciate her attention and valuable time that contributed in completing this thesis successfully.

Second, my deep appreciation to my family especially my parents for the sacrifice they offered to me to reach this level of higher education and confidence in my practice.

I would also like to extend my thanks to my wife and daughters who helped me to carry out this thesis by their patience. Their continuous support has sustained my efforts at all stages of this thesis. They assist me in accomplishing this thesis by their motivation and inspiration.

ABSTRACT

DEVELOPMENT OF A FRAMEWORK FOR FREQUENT ITEMSET MINING UNDER MULTIPLE SUPPORT THRESHOLDS

Frequent pattern mining is an essential method of data mining that is used to extract interesting patterns from massive databases. Traditional methods use single minimum support threshold to find out the complete set of frequent patterns. However, in real word applications, using single minimum support threshold is not adequate since it does not reflect the nature of each item and causes a problem called rare item problem. Recently, several methods have been studied to tackle this problem by avoiding using single minimum item support threshold. The nature of each item is considered where different items are specified with different minimum support thresholds. By this, the complete set of frequent patters are generated without creating uninteresting patterns and losing substantial patterns. In this thesis, we propose an efficient method, Multiple Item Support Frequent Pattern growth algorithm, MISFP-growth, to mine the complete set of frequent patterns with multiple item support thresholds. In this method, Multiple Item Support Frequent Pattern tree, MISFP-Tree, is constructed to store all crucial information to mine frequent patterns. Since in the construction of the MISFP-Tree is done with respect to minimum of Multiple Itemset Support values; pruning and reconstruction phases are not required. To show the efficiency of the proposed method, it is compared with a recent tree-based algorithm, CFP-growth++. To evaluate the performance of the proposed algorithm, various experiments are conducted on both real and synthetic datasets. Experimental results reveal that MISFP-growth outperforms the previous algorithm in terms of execution time, memory space as well as scalability.

ÖZET

ÇOKLU DESTEK EŞİKLERİNDE SIK KÜMELER MADENCİLİĞİ İÇİN UYGULAMA İSKELETİ GELİŞTİRİLMESİ

Sık kümeler madenciliği yöntemleri yoğun veri tabanlarındaki özellikli örüntülerin bulunmasını sağlarlar. Bu yöntemler, sık kümeler setlerini bulurken tek bir destek eşik değerini esas alırlar. Oysa gerçek dunya uygulamalarında tek bir destek eşik değeri örüntülerin tek başlarına özelliğini yansıtmakta yetersiz kalmakta ve seyrek örüntü (rare item) problemi ortaya çıkarmaktadır. Son zamanlarda, bu seyrek örüntü probleminin çözümüne odaklanan çalışmalar bulunmaktadır. Bu çalışmalar kümelere ve küme elemanlarına farklı destek eşik değerleri atanmasına izin vermektedir. Böylece gereksiz örüntüler oluşturulmadan seyrek örüntüler ele geçirilebilmektedir. Bu tez kapsamında, etkin bir Çoklu Destek Eşiklerinde Sık Kümeler (Multiple Item Support Frequent Pattern growth algorithm, MISFP-growth) yöntemi önerilmektedir. Bu yöntem veri tabanından sık örüntülerin bulunmasını sağlayacak veriyi bir ağaçta saklamaktadır (MISFP-Tree). Bu ağaç tüm veriyi değil de çoklu eşiklerin minimumunu dikkate alarak oluşturulduğu için, oluşturulma sonrası budama ve yeniden oluşturulmaya gereksinim duymamaktadır. Bu yöntemin etkinliği yeni bir ağaç tabanlı yöntemle (CFP-growth++) karşılaştırılarak gösterilmiştir. Karşılaştırma çalışmaları gerçek ve sentetik veri tabanları üzerinde gerçekleştirilmiştir. Başarım değerlendirme sonuçları MISFP-growth yönteminin diğer yönteme göre, çalışma zamanı, bellek kullanımı ve ölçeklenebilirlik açısından daha başarılı olduğunu göstermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

Data mining is defined as "the analysis of the large quantities of data that are stored in computers" [29]. It is a computational process to find out interesting information from masses of data. It is used in various domains such as: medical field, elections, telecommunication firms, banks [30, 34]. The methods in data mining include association rule mining [35], clustering [36, 37] and classification [38].

Association Rule Mining (ARM) is an important pattern analysis technique that focuses on finding out the sequences of actions or events known as itemsets (patterns) that are items that occur together. It has drawn attention since it was first proposed in [1]. This is due to its applicability in various domains. The overall goal of ARM is to discover all interesting rules from a dataset that have the form: $X \rightarrow Y \mid X \cap Y = \emptyset$ where X and Y are the set of items in the dataset. An interesting rule should satisfy two statistical measures known as minimum support threshold denoted as *minsup* and minimum confidence threshold denoted as *minconf*. *Minsup* refers to the percentage of transactions in the dataset that contain $X \cup Y$ whereas *minconf* denotes to the conditional probability of finding $X \cup Y$ given the transactions of which each contains X.

Market Basket Analysis is an application of association rule mining. In this application, relationship between items that purchased together is analyzed. Customers who purchase bread also tends to purchase milk at the same time can be represented by association rule: milk ⇒ bread [sup= 30%, conf = 75%]. A support of 30% for the previous association rule means that 30% of all transactions in database contain bread and milk whereas a confidence of 75% for the previous association rule shows that 75% of the customers who buy bread also tends to buy milk.

Association rules can be found in two essential steps as follows.

1. Finding all frequent patterns that exceeds a given *minsup*.
2. Generating association rules. From the frequent patterns that are found in step 1, we extract the all interesting rules that satisfy both of *minsup* and *minconf*.

Since the first step is more expensive, almost all research in frequent pattern mining algorithms focused on generating the frequent patterns. Also, once the frequent

patterns are generated, generating association rules is straight forward since confidence does not possess closure property as support. Hence, Frequent Pattern Mining (FPM) is essentially focuses on finding out items that occur together (known as patterns or itemsets).

There are various methods proposed to find out frequent patterns in large databases. Initially, a brute-force approach (primitive method) is used to find out all frequent patterns [35, 39]. The main idea behind this method is that all possible itemsets are generated. Then, all itemsets that satisfy a given minimum support threshold are considered frequent patterns. Although this approach has guarantee to find out all frequent patterns, it requires O(NMw) comparisons, where N is the number of transactions, M =2k-1 is the number of candidate itemsets, w is the maximum transaction width and k is the number of distinct items. Hence, using this approach is very expensive [39].

To overcome a combinatorial explosion, Apriori principle has been proposed in [2] to prune itemsets contain at least one infrequent subset itemset. Subsequently, many methods have been proposed in order to improve the implementations of the first proposed algorithm [1] as Apriori-like methods [2, 44, and 45], depth-first search methods [46], vertical methods [40, 41, and 42], node-based methods [25, 27] and FP-growth-like methods [3, 5, 6, 26, and 7]. All these methods share the same aim as to reduce both of search space and execution time. A single *minsup* is used to extract the whole set of frequent patterns. But using a single minsup in real-life applications does not reflect the importance of individual items since mining with a single *minsup* implicitly assumes that all items in the database are of the same nature in the database. In the real-life applications, some items are bought frequently whereas others are bought very rarely. Cheaper goods such as, milk and bread, are frequently purchased while the luxury and expensive products such as, electric devices are infrequently purchased. Although those infrequent items they may create more profit than frequent items. Hence, finding rare itemsets is desired but without generate large amount of meaningless itemsets.

To generate a rare itemsets, *minsup* must be set too low, in this case too many meaningless itemsets are generated. In contrast, if *minsup* is too high, lots of useful itemsets are lost. This problem is called rare itemset problem [23]. To overcome this problem, a numerous methods have been proposed to discover rare itemsets. In [8], instead of using a single *minsup* for all items, each item can have its minimum item support to precisely reflect the accurate nature of each item in database. By this, both of

rare itemsets and frequent itemsets are generated. Thus, mining itemsets with *Multiple minimum Item Supports thresholds*, MIS, effectively discovers the frequent itemsets as well as rare items without generating a huge amount of useless frequent itemsets and losing desired rare itemsets become possible.

Since MSapriori [8], many methods have been proposed to reduce search space and execution time while generating frequent patterns under MIS. These methods discover all meaningful rules with MIS, containing rarely occurred by applying different MIS to each different items. They can be classified into two types: 1) Apriori-like methods [4, 8, 9, 10, 19, 20, and 22] and 2) FP-growth-like methods [11, 12, 13, 14, 15, 17, 18, and 16]. In Apriori-like methods, the databases have to be scanned many times to create all candidates items since they are based on Apriori algorithm. Apriori-like methods consume an enormous amount of runtime, especially when databases contain too many transactions since it performs several database scans and uses a candidate generation-and-test method. To solve this problem, FP-growth-like methods were proposed. These methods require scanning database twice as they use FP-Tree to hold all necessary information that is needed in mining process. They construct MIS-Tree with a single scan and the MIS-Tree is reconstructed by pruning operations. However, these methods still consume a huge amount of memory space and execution time. They are poor in performance since they require pruning and reconstructing the MIS-Tree.

In this thesis, we propose an efficient algorithm called Multiple Item Support Frequent Pattern growth, MISFP-growth, which is an extended version of FP-growth, for mining frequent patterns with MIS over big databases. The proposed algorithm is intended to create all potential itemsets from huge transactional datasets. We propose Multiple Item Support Frequent Pattern tree, MISFP-Tree, which is based on FP-Tree, to hold all necessary information to discover the complete set of frequent patterns with MIS. At the same time, a frequent item header table, MIN-MIS-Frequent table, is generated with all items that have support no less than the lowest minimum item support threshold, MIN-MIS. The proposed tree is efficiently built without any reconstruction or pruning operations. Then, the proposed method, MISFP-growth, extracts frequent patterns from MISFP-Tree. The experimental results on both real and synthetic datasets show the superiority of the proposed method in terms of runtime and memory with respect to varying minimum item support thresholds and database sizes. Major part of this work has been presented in Wseas 16th International Conference on Applied Computer Science and it is published in its proceedings, WSEAS Transactions on Computer Research [61].

To assess the efficiency of MISFP-growth in term of execution time, memory space and scalability, experiments are carried on both real and synthetic datasets that are commonly used in the data mining literature. In these experiments, we compare the performance of the proposed algorithm with the recent based-tree algorithm, CFP-growth++, based on multiple minimum item support thresholds [14]. To evaluate the execution time and memory usage of both algorithms, we vary MIS of items and number of transactions. The experimental results show that the proposed algorithm, MISFP-growth, is faster, consumes less memory and is more scalable. Most of this work has been published in [61].

This thesis is organized as follows; Chapter 2 focuses on introducing several concepts and definitions about mining frequent patterns. Chapter 3 provides a review about the rare item problem and the up-to-date methods that are used to discover frequent patterns under multiple support thresholds. Chapter 4 introduces the proposed method called, MISFP-growth that tackles the rare item problem by mining frequent patterns with multiple minimum support thresholds. Chapter 5 gives the results of experimental set up of MISFP-Growth algorithm and finally Chapter 6 provides a summary about the proposed method and remarks on future work.

In this thesis, the terms method and algorithm; itemset and pattern as well as dataset and database are used interchangeably

CHAPTER 2

PRELIMINARIES

In this chapter, we introduce the basic terminologies related to frequent pattern mining under both of single and multiple thresholds.

Let $I = \{i_1, i_2, \dots, i_m\}$ represents the set of m distinct items, and $DB = \{T_1, T_2, \dots, T_n\}$ be a transaction database where T_i $(i \in [1 \dots n])$ is a transaction, which contains a set of items in *I*. Each transaction is associated with an identifier, called *TID*. A transaction can be defined as $T_i = (TID_i, X)$, which is a tuple has number *TID* and contains an itemset *{X}*. The itemset $X = \{x_1, x_2 \dots x_k\}$ is a set of k items in T. Thus, the itemset *{X}* have at least one item and at most all items in specific transaction. The itemset that contains *K* items is called *K*-itemset. If support of the itemset is greater than or equal to *minsup***,** then it is a frequent pattern. The support of the itemset *X*, denoted as sup*(X)*, is the number of transactions that contain *{X}* in *DB* as follows:

$$sup(X, DB) := \{TID | (TID, I) \in DB, X \subseteq I\}. \quad (2.1)$$

Definition 2.1: (Frequent Pattern with Single Threshold). Let *DB* be a transaction database over a set of items *I*, and *minsup* is a given minimal support threshold given by the user. The set of frequent patterns in *DB*, which they exceed *minsup* is defined as follows:

$$F(DB, minsup) = \{\mathrm{X} \subseteq \mathrm{I}, \mathrm{sup}(\mathrm{X}, \mathrm{DB})/|DB| \geq minsup\}, \quad (2.2)$$

where *F* represents all the frequent itemsets in *DB* and $|DB|$ is the number of transactions in *DB*.

An association rule is represented as an expression of the form $X \Rightarrow Y$ in *DB*, where$(X, Y \subseteq \mathrm{I}$ and $X \cap Y = \theta)$. Itemsets *X* and *Y* are called antecedent and consequent of the rule respectively. The two statistical measures (support and confidence) are used to determine the importance of an association rule. The association rule's support is represented by the union of its antecedent and consequent whereas the confidence of a rule represent by the conditional probability of consequent *(Y)* given antecedent *(X)*. Hence, the support (sup) and confidence (conf) of a rule $X \Rightarrow Y$ is as follows.

$$\mathrm{sup}\left(X \Rightarrow Y, DB\right) = \frac{\mathrm{sup}(X \cup Y)}{|DB|}, \quad (2.3)$$

$$conf\left(X \Rightarrow Y, DB\right) = P(X) = (\sup(X \cup Y))/(\sup(X)). \quad (2.4)$$

Definition 2.2: Association Rule (*AR*). Let I be a set of items, *DB* be a transaction database *I*, *minsup* is a user-specified minimal support threshold, and *minconf* is a user-specified minimal confidence threshold. The set of interesting association rules that satisfy both of *minsup* and *minconf* defined as follows:

$$AR(DB, minsup, minconf) \coloneqq \left\{X\ (\Rightarrow Y | X, Y \subseteq \mathrm{I},\ \ \mathrm{X} \cap \mathrm{Y} = \theta, \mathrm{X} \cup \mathrm{Y} \in \mathrm{F}, \mathrm{conf}\left(\mathrm{X} \Rightarrow\right) Y, DB) \geq minconf\right\}. \quad (2.5)$$

The following example illustrates the concepts of Frequent Patterns (*FP*) and Association Rule (*AR*).

Example 2.1: Let I be a set of items *{a, b, c, d, e, f, k}*, and a set of transaction database *DB* over *I* showed in Table 2.1, find out all frequent pattern *F* (*DB*, *minsup* =50%) and AR (*DB*, *minsup* = 50%, *minconf* = 60%). The complete set of frequent patterns and their supports can be obtained in Table 2.2.

Table 2.1. Transaction database

TID	Items
1	a ,b, c ,d, e
2	a, b, e
3	c, d, f
4	b, c, k
5	a, b, c, d

Table 2.2. Frequent patterns with single threshold

Patterns	Support	Frequency	Status
{}	5	100%	Frequent
{a}	3	60%	Frequent
{b}	4	80%	Frequent
{c}	4	80%	Frequent
{d}	3	60%	Frequent
{a, b}	3	60%	Frequent
{c, d}	3	60%	Frequent
{b, c}	3	60%	Frequent

For association rule $\{c\} \Rightarrow \{d\}$, we can get the support and confidence of the rule as follows.

$$\sup\left(\{c\} \Rightarrow \{Y\}\right) = \sup(\{c\} \cup \{d\}) = 60\%,$$

$$conf\left(\{c\} \Rightarrow \{d\}\right) = \frac{\sup(\{c\} \cup \{d\})}{\sup(\{c\})} = \frac{60\%}{80\%} = 75\%.$$

Property 2.1: Downward Closure Property. All subset of frequent itemset must be frequent, in other words, an infrequent itemset will not generate any frequent itemset. For example, itemset *{a, e}* is not frequent itemset since it contains infrequent subset itemset, *{e}*.

Definition 2.3: Rare Itemset (*RI*). Let *DB* be a transaction database over a set of items *I*, and *minsup* is a given minimal support threshold. The set of rare itemsets in *DB*, which have support bellow than *minsup* is defined as follows:

$$RI(DB, minsup) = \{\mathrm{X} \subseteq \mathrm{I}\,, \sup(\mathrm{X}, \mathrm{DB})/|DB| < minsup\}, \qquad (2.6)$$

where *RI* represents all the rare itemsets in *DB* and $|DB|$ is the number of transactions in *DB*. For example itemset *{b, e}* is rare itemset since its support = 40%, which is less than *minsup* = 50%.

Definition 2.4: Multiple Item Support (*MIS*). Let *I* be a set of items, $I = \{i_{1,\ldots}, i_n\}$, an itemset $X = \{i_1, \ldots, i_k\}$, the minimum item support(*MIS*) of itemset *X* is defined as follows.

$$MIS(X) = \mathrm{MIN}\{\mathrm{MIS}(i_1), \mathrm{MIS}(i_2), \ldots, \mathrm{MIS}(i_k)\}. \qquad (2.7)$$

Example 2.2. Assume that an itemset *K={x, y, z}* has an actual support = 8% in a given database. Suppose that the *MIS* and the actual support of items are given as follows:

$$MIS(x) = 5\%, MIS(y) = 10\% \; and \; MIS\;(z) = 15\%,$$

$$sup(x) = 10\%, sup(y) = 9\% \; and \; sup\;(z) = 11\%.$$

Then the MIS of the itemset K can be defined as follows:

$$MIS(K) = MIN\;\{MIS(x) = 5\%, MIS(y) = \;10\%, MIS\;(z) = 15\%\} = \;5\%.$$

Thus, the itemset *K* is frequent as its support = 8%, which exceeds *MIS (K) = 5%*.

Definition 2.5: (Frequent Itemset with Multiple Thresholds). Let *DB* be a transaction database over a set of items *I*, and *MIS* is specified for each item. The set of frequent patterns with *MIS* in *DB*, is as follows:

$$MSF(DB, \mathrm{MIS}) = \{\mathrm{X} \subseteq \mathrm{I}\,, \sup(\mathrm{X}, \mathrm{DB}) \geq \mathrm{MIS(X)}\}, \qquad (2.6)$$

where MSF represents all the frequent itemsets.

Example 2.3. Consider the database in Table 2.1 and the minimum item supports of items are given as follows:

MIS(a) = 30%, MIS(b) = 25%, MIS(c) =30%, MIS(d) = 40%, MIS(e) = 25%, MIS(f)= 30% and MIS(k)=25%. The complete set of frequent patterns with MIS can be obtained in Table 2.3.

Table 2.3. Frequent patterns with multiple thresholds

Patterns	Support	Frequency	MIS	Status
{a}	3	60%	30%	Frequent
{b}	4	80%	25%	Frequent
{c}	4	80%	30%	Frequent
{d}	3	60%	40%	Frequent
{e}	2	40%	25%	Frequent
{a, b}	3	60%	25%	Frequent
{a, c}	2	40%	30%	Frequent
{a, d}	2	40%	30%	Frequent
{a, e}	2	40%	25%	Frequent
{b, c}	3	60%	25%	Frequent
{b, d}	2	40%	25%	Frequent
{b, e}	2	40%	25%	Frequent
{c, d}	3	60%	30%	Frequent

For the itemset {a, f}, it is infrequent itemset as its support = 20%, which is less than its MIS = MIN {MIS (a), MIS (f)} = 30%.

Note that although the itemset {b, e} not appear while mining patterns under single threshold and it considered as rare itemset, it is appear as interesting pattern when we mine patterns with MIS. Thus, mining frequent patterns with multiple thresholds efficiently discovers interesting patterns, which include frequent and rare patterns.

CHAPTER 3

RELATED WORK

Data mining is a nontrivial process of discovering interesting knowledge from big databases. Frequent pattern mining is an essential step of data mining, which aims at discovering a set of itemsets that frequently co-occur in a database. Frequent itemsets should satisfy user-specified minimum support threshold, *minsup.* Since the first method was introduced to extract frequent patterns in [1], there are numerous algorithms that have been proposed for mining frequent patterns under single threshold. Most of these methods utilize the *downward closure* property to considerably reduce the search space of frequent patterns [2, 3, 5, 6, 7, 40, 41, 44 and 45]. In these methods, the database is scanned to determine all frequent items (or *1*-itemsets) based on the single *minsup* threshold. Then, only data of frequent items are used to determine the frequent *itemsets* (i.e. frequent patterns). Thus, these methods significantly decrease the memory space and execution time by avoiding a large amount of infrequent data from being loaded into memory.

Frequent itemset mining methods can be classified into two main categories: Apriori-like methods [1, 2, 35, 54, 55, 56, and 57] and FP-growth-like methods [3, 5, 6, 7, and 60]. Apriori methods deploy a breadth-first search to count the support of (k+1)-itemsets that are created from frequent k-itemsets. It achieves good performance by reducing the search space as the *downward closure* property is utilized. Since it is considered an innovation that opened new doors for many frequent pattern mining applications, many variants of Apriori [2] have been proposed to enhance the performance of Apriori such as BitApriori [54], Hybrid Search Based Association Rule Mining [55], Direct Hashing and Pruning (DHP) [56], and Dynamic Itemset Counting (DIC) [57], etc. On the other hand, the multiple database scan approach of Apriori algorithm is I/O expensive for large databases. In addition, due to the candidate generation-and-test approach, it requires huge computational time and memory usage when too many candidate itemsets are generated.

To handle these weaknesses, FP-growth [3] and its improvements [5, 6, 7, 26, and 60] have been proposed to generate frequent patterns without creating a huge amount of candidate itemsets as Apriori. FP-growth methods utilizes FP-Tree, an extended prefix-tree, which compresses all transactions of database in horizontal data format in memory.

This enables FP-growth to search for the complete set of frequent patterns and to reduce the number of database scans. FP-growth outperforms previously developed methods including Eclat [59] and Apriori methods [2, 53, 54, 55, and 56]. However, a huge amount of frequent patterns are generated while mining from some dense databases and mining with low minimum support threshold. In addition, for each frequent k-itemset, FP-growth creates set of conditional FP-Tree used to find the frequent (k+1)-itemsets. This leads to create a large number of FP-Trees which results in the degradation of performance. Thus, FP-growth does not work as well as Eclat [59], in such cases. The extensions of FP-growth have been proposed to enhance the performance that includes an array technique to reduce the FP-Tree traversal time [58], FP-growth with database partition projection [60], H-mine [52] and the use of FP-array data structure [28].

Many different data structures are used to represent databases in memory such as Matrix Apriori [53], MAFIA [51], FP-Tree [3], etc by avoiding re-scanning the original DB from disks as in the Apriori method [2]. These data structures and their mining methods are quite different which result in different performance for a given dataset. Methods like Apriori [2], FP-growth [3] and H-mine [52] exploit horizontal format of data while Eclat [40] and MAFIA [51] present data in vertical format. Thus, applying a suitable mining method for mining frequent patterns is crucial to improve the performance of frequent pattern mining.

Above methods are used to find frequent patterns with single *minsup*. Using the single *minsup* considerably reduces the search space and computation by avoiding a huge amount of infrequent itemsets from being loaded into memory. However, mining frequent patterns with the single *minsup* faces two problems: 1) extremely large amount of meaningless patterns might be generated if the *minsup* is set too low, 2) useful patterns may be lost when the *minsup* is set too high. This problem is called rare item problem.

The remaining of this chapter is divided into two subsections. In first subsection, we briefly discuss the rare item problem. In the second subsection, we review the methods that have been proposed to mine frequent patterns under multiple support thresholds.

3.1 Rare Item Problem

In many applications, frequent items are less interesting than rarely occurred ones since frequent patterns signify the known and the expected associations while rare patterns suggest unexpected or previously unknown associations, which is more valuable to users [48]. A rare itemset contains items that have support less than *minsup*, which are called rare items. These items can be obtained by setting *minsup* at a low value but this leads to combinatorial explosion in number of frequent itemsets. Thus, the problem of specifying single support threshold causes rare item problem [8]. By mining frequent patterns with a high *minsup,* we cannot find itemsets that contain useful rare items while mining with a low *minsup* generates large amount of frequent patterns which are not useful. Many methods have been proposed to tackle this problem [24, 23, 47, and 48]. The main effort of these methods is discovering rare itemsets.

In [24], two automated support thresholds are used to mine rare itemsets. These support thresholds can be calculated as follows. One support threshold represents the average support of all items, which is denoted as *AvgSup*, and the second support threshold can be found as the average of the lowest support and large support, which is called *MedianSup*. According to these support thresholds, this method mines frequent and rare itemsets belonging to three different item groups namely Most_interesting_ Group (MiG), Somewhat_interesting_Group (SiG), Rare_interesting_Group (RiG). These groups consist of itemsets which have support greater than AvgSup, between *MedianSup* and *AvgSup* and less than *AvgSup* and *MedianSup* respectively. Thus, the rare itemsets are found by scanning the itemsets in RiG.

In [23], two methods have been proposed to extract the rare itemsets. In these methods, three type of itemsets are defined: minimal generators (MG), minimal rare generators (MRG), and minimal zero generators (MZG). They represent itemsets with lower support than its subsets, itemsets with non-zero support and whose subsets are all frequent and itemsets with zero support and whose subsets all have non-zero support respectively. MRGs play an important role in these methods since they represent a boundary that separates the frequent and rare itemsets. In addition, MRG is used by MRG-Exp method to create candidates in bottom-up fashion by using MGs. MRGs is utilized by the second method, called ARIMA, to create the whole set of rare itemsets which was

created in the first method. This method stops the search for non-zero rare itemsets when MZG reaches to border since above that boundary there are only no rare itemsets.

In [47], RP-Tree is proposed to mine rare itemsets using a tree structure. The aim of this method is to find out rare itemsets using a tree structure. It is the first rare association rule mining algorithm that uses a tree structure. It is a modification of the FP-Growth algorithm. Like FP-growth method, this method scan database twice. In the first scan, it calculates the actual supports of items whereas in the second scan it builds initial tree. RP-Tree is used to store the transactions that have at least one rare item. By this, all the transactions that have non-rare item are not included in RP-Tree construction. This method uses tree data structure to extract rare itemsets.

The improvement method of RP-Tree [48], called MCRP-Tree, was proposed to mine rare itemsets. The main aim of this method is to find rare itemsets using tree structure with maximum constraint model. RP-Tree is level-wise approach which generates and tests all combinations of rare itemsets, whereas MCRP-Tree does not test all items containing in the rare itemset. Instead the MCRP-Tree discovers only rare items from the transactional dataset. It avoids expensive pruning step and item generation by using tree data structure based on FP-Tree to find rare items. It focuses on rare itemsets which gives interesting rule and does not spend time in finding uninteresting rules. MCRP-Tree contains only rare items by discarding all the transactions that does not have rare items.

3.2 Mining Frequent Patterns with Multiple Thresholds

Recently, to tackle the rare item problem, several methods have been proposed to extract frequent patterns involving rare itemsets with multiple support thresholds [8, 9, 10, 11, 12, 13, 14, 19, and 22]. In these methods, the definition of minimum item support is changed while the definition of association rule remains the same while mining frequent patterns with MIS. Each item is assigned by MIS to reflect the nature of that item in a database. The main idea behind these methods is that any itemset must satisfy the lowest MIS of its items. Thus, these methods enable us to find interesting patterns including rare patterns without creating a huge number of meaningless patterns.

To find out frequent patterns with single *minsup* threshold, *downward closure* property (an itemset is frequent if and only if all its subsets are frequent) must hold. In [8], it has be shown that this property no more holds for mining with MIS and they

proposed a new property called *sorted closure* property to overcome this problem. The next example 3.1 illustrates why the *downward closure* property cannot hold to mine frequent patterns with MIS.

Example 3.1. Consider four items {a, b, c, d} in a database and their minimum item supports are:

MIS(a) = 15%, MIS(b) = 25%, MIS(c) = 10%, and MIS(d) = 9%.

Assume that the support of {a, b} is 10% at level 2, the itemset's support is not greater than either MIS (a) or MIS (b). Thus, {a, b} is discarded since it is infrequent itemset according to the definition of mining patterns with MIS. Therefore, {a, b, c} and {a, b, d} are infrequent patterns since they include {a, b}. Suppose we find that itemset {a, b, c} has 12% of support at level 3, then it satisfies: min {MIS (a), MIS (b), MIS(c)} since the MIS(c) is less than the actual support of itemset. As result, {a, b, c} is considered as desired pattern although its subset {a, b} is infrequent. Thus, *downward closure* property is no more valid to find frequent patterns with MIS instead a new property proposed to solve this problem called sorted closure property.

Definition 3.1 Sorted Closure Property. Items *I* must be sorted in ascending order according their MISs. Assume that L_k denote to set of K-itemsets, any itemset x consist of k items $\{ x_1, x_2, \ldots, x_k\}$, then $MIS(x(1)) \leq MIS(x(2)) \leq \ldots \leq MIS(x(k))$.

Example 3.2. Following the above example assume that an itemset, k = {a, b, c}. The items that is contained in this itemset must be sorted in ascending order according to their MIS. Therefore, K ={c, a, b}.

In the following subsections, we will discuss these methods that can be divided into two parts: Apriori-like methods and FP-growth-like methods.

3.2.1 Apriori-Like Methods

To avoid the rare item problem, an algorithm called MSapriori [8] is proposed to find out frequent patterns with multiple item support thresholds. It is an extension of Apriori algorithm, which is used to discover frequent itemsets involving rare items. This method assigns a *minsup* value known as MIS for each item. Frequent itemsets are found if an itemset satisfies the lowest MIS value among the respective items. In this method, the frequent items are assigned with a higher MIS value whereas rare items are assigned

with a lower MIS value. Thus, the MSapriori algorithm addresses the rare itemset problem and improves the performance over single *minsup* based algorithms.

The algorithm MSapriori works as Apriori algorithm with some modification in the second candidate itemsets generation as follows:

1- The database is scanned once to count the actual support of items and to determine first-frequent patterns (L_1).

2- In each subsequent scan, it generates k-candidates itemsets by using the seeds of frequent patterns found in L_{k-1} and it creates L_k frequent patterns based on C_k candidate's items, this step repeats tell no more candidate's itemsets can be generated. There is an exception in the second step as we will see later in the following example.

3- All items are sorted in ascending order according to their MIS values and all subsequent operations of this method used this order.

4- An itemset X: $\{a_1, a_2, \ldots, a_n\}$ is frequent if and only if its actual support in the database is no less than to: min $\{MIS(a_1), MIS(a_2), \ldots, MIS(a_n)\}$.

To understand how this method works to find out the whole set of frequent patterns, the following Example 3.3 illustrates all steps involved in this algorithm.

Example 3.3. Given a transaction database DB as shown in Table 3.1 and the multiple item supports of items in Table 3.2, find the whole set of frequent patterns with MIS in DB.

To find the frequent patterns from the data in Table 3.1 with the multiple predefined minimum support values in Table 3.2, the MSapriori proceeds as follows.

1. Scan the database once to find the actual support of each item. The items are sorted in ascending order according to their MIS as shown in the right column in Table 3.1 and the actual support of items is counted as shown in the third row in Table 3.2.
2. After we get the frequencies of all items, we find out the 1-candidates itemsets that have support greater or equal to the MIS' value of the first item = 2. Thus, The items {g, f, c, b, a} form F' list that used to create the 2-candidate itemsets as special stage of MSapriori.
3. The 1-frequent patterns *{g, c, f}* are generated since their MIS values are greater or equal to 2 and their actual support is greater or equal to their predefined MIS. Therefore, $L_1 = \{g, c, f\}$. You can see that items {a, b} are removed since their support in database less than their MIS.

4. Find the 2-candaidate itemsets is an exception for another steps since it will be involve items found in F to create 2-candidate patterns. Thus, the 2-candiadate itemsets are created from F' list instead of L_1. All the 2-candiataes itemsets are given in second row in the Table 3.3, where the number after ":" represents the actual support of the itemsets. Then , the 2-frequent patterns L_2= *{ (g, f) : 2 , (f ,c) :3 , (f, b) : 2 , (f, a) :3 }* are generated.
5. The same process is repeated to generate all frequent pattern L_k based on candidate itemsets in C_k. by repeating the same steps tell no more candidate itemset can be created, all candidate itemsets and the complete set of frequent patterns are found as shown in Table 3.3.

Table 3.1. Transaction database

TID	Items	Items in ascending order in MIS values
1	d, c, a, f	f, d, c, a
2	g, c, a, f, e	g, f, e, c, a
3	b, a, c, f, h	h, f, c, b, a
4	g, b, f	g, f, b
5	b, c	b, c

Table 3.2. MIS and the actual support of items

Item	a	b	c	d	e	f	g	h
MIS	4	4	4	3	3	2	2	2
Actual support	3	3	4	1	1	4	2	1

Table 3.3. The whole frequent patterns using MSapriori

Step no	Candidate Itemsets	Frequent Patterns	Notes
1	*F={g :2, f:2, c:4 , b: 3, a :3}*	*L_1={ g :2, c :4, f:2 }*	
2	*C_2={ (g, a) : 1 , (g, b) : 1, (g, c) :1, (g, f) : 2,(f ,c) :3 , (f, b) : 2 , (f, a) :3, (c, b) :1, (c, a) : 3, (b, a) :1 }*	*L_2={ (g, f) : 2 , ,(f ,c) :3 , (f, b) : 2 , (f, a) :3 }*	C_2 created from F
3	C_3={(f, c, b) : 1,(f, c, a) : 3}	L_3={f, c, a}	

Several studies have been proposed to improve the performance of MSapriori [9, 10, 19, and 22]. In [9], an MSB_apriori method was proposed to find frequent itemsets with MIS based on Apriori algorithm. In this method, each item in the database has its own MIS determined by the user and each itemset can satisfy a different *minsup* based on the items within it. Unlike MSapriori which uses different steps from Apriori algorithm to mine frequent patterns with MIS, MSB_apriori uses the same steps involved in basic

Apriori to find frequent patterns. MSB_apriori has two main steps as follows: 1) mine frequent patterns L by basic Apriori with a single minimum support, 2) choose all frequent patterns from L that satisfy the definition of frequent patterns with multiple minimum supports from L. So, any itemset F (F∈L) must satisfy sup (F) ≥ MIS (1) to be frequent since items are in ascending order according their MIS.

In [10], another method have been proposed to mine frequent patters with MIS called an Improved Multiple Support Apriori Algorithm (IMSApriori). This method is different from MSapriori in the approach of specifying *minsup* for each item. The Support Difference **(SD)** is used to assign the minimum item support thresholds to items and it refers to the acceptable deviation of an item from its frequency so that an itemset involving that item can be considered as a frequent itemset. The following formula is used to calculate the *minsup* known as MIS (i) for each item 'i' as follows:

$$MIS(i) = \begin{cases} S(i) - SD & if (S(i) - SD) > LS \\ LS & othewise \end{cases} \quad (3.1)$$

where, S (i) denotes the support of item 'i' and LS refers to the given least support.

The support difference SD can be calculated as in following equation:

$$SD = \lambda\ (1-\alpha) \quad (3.2)$$

where λ represents the parameter maximum support of the item supports and **α** is the parameter ranging between 0 to 1 and SD values changed from (0, λ). It efficiently decreases the explosion of frequent itemsets including frequent items without affecting the extraction of frequent itemsets involving rare items.

In [19 and 22], two methods have been proposed to mine frequent patterns with MIS. These methods work as MSapriori with the some differences as follows. In [19], it first finds all the frequent 1-itemsets for the given database by comparing the support of each item with its predefined minimum support. It then finds all the frequent k-itemsets for the database by comparing the support of each candidate k-itemset with the maximum of the minimum supports of the items contained in it. In [22], all the steps that are used in this method are same as that used in MSapriori with following exception. The minimum item support thresholds for each item, *i*, can be calculated by the following equation:

$$MIS(i) = \begin{cases} \beta\ s(i) & \beta s(i) > LS \\ s(i) & else \end{cases} \quad (3.3)$$

where β , s(i) and LS stand for a user-specified value which can be ranged from 0 to 1, support of an item (i), and the least minimum support threshold respectively .

This method uses the above equation to ensure the extraction of frequent itemsets involving rare itemsets. Furthermore, it prunes frequent itemsets involving frequent items in a more efficient manner and without missing the frequent itemsets involving rare items.

3.2.2 FP-growth-Like Methods

The above methods adopt an Apriori-like candidate set generation-and-test approach and it is always time consuming, especially when there exists long patterns. So, a novel multiple item support tree (MIS-Tree) structure [12], which extends the FP-tree structure [3], is proposed for storing compressed and crucial information about frequent patterns. MIS-Tree-based mining method (CFP-growth algorithm) is also developed for mining the complete set of frequent patterns with multiple support thresholds.

The CFP-growth algorithm is designed based on the FP- growth algorithm but it is different from FP-growth in the structure, construction and mining procedures. The CFP-growth algorithm receives as an input both of MIS values of items and transaction database. It finds out the whole set of frequent itemsets with a single scan of the transaction database. The CFP-growth has four phases as follows.

1- Items are sorted in decreasing order of their MIS values. Using the sorted list of items, an FP-Tree-like structure known as MIS-Tree is built with a single scan of the transaction database. Concurrently, the frequency of each item in the MIS-Tree is counted.
2- To discard items that cannot create any frequent pattern, a tree-pruning operation is carried out to prune them.
3- After some items are removed from MIS-Tree by the tree-pruning operation, some items in the tree may have child nodes with the same name. Therefore, to merge the child nodes of a parent node that share the same item, the tree-merging operation is carried out on the MIS-Tree to merge them. After tree-pruning and tree-merging operations, the compact structure will be formed called a compact MIS-Tree, which holds the substantial information that is needed to mine frequent itemsets.
4- Lastly, to discover the whole set of frequent patterns form the compact MIS-Tree ,each item is selected from the compact MIS-Tree as the suffix item (or itemset),

its conditional pattern base (i.e., prefix sub-paths) is constructed to generate the complete set of frequent itemsets. Due to mining frequent itemsets with MIS does not satisfy *downward closure* property, CFP-growth attempts to find out a complete set of frequent itemsets by building suffix patters till its respective conditional pattern base is empty.

Example 3.4. Consider the transaction database given in Table 3.4 and the MIS values of items in Table 3.5, find the whole set of frequent patterns using CFP-growth algorithm.

Table 3.4. Transaction database

TID	Items	Items in descending order in MIS values
1	d , c , a , f	a, c, d, f
2	g , c , a ,f , e	a, c, e, f, g
3	b , a , c , f, h	a, b, c, f, h
4	g , b ,f	b, f, g
5	b , c	b, c

Table 3.5. MIS values of items

Item	a	b	c	d	e	f	g	h
MIS	4	4	4	3	3	2	2	2

Constructing of MIS-Tree phase

The database has to be scanned once to calculate the support of items in DB and to build MIS-Tree. Similar to FP-Tree construction, transactions are inserted into MIS-Tree with an exception that here items are in descending order according their MIS. By adding all transaction, the MIS-Tree is created as shown in the Figure 3.1.

MIN-frequent-header table is constructed to facilitate MIS-Tree traversal, each item in this table points to items that carry the same item-name via the head of node-links. The supports of items after scanning all transactions are ((a:3, b:3, c:4, d:1, e:1, f:4, g:2, h:1). Thus, the pruning operation is implemented to discard items that have actual support less than the lowest MIS = 2. So, the items (d, e, h) are pruned from the MIS-Tree. After pruning operation, some nodes may have child nodes carrying the same name and the child nodes have to be merged to one node with the same name and its count is set as the sum of merged child nodes.

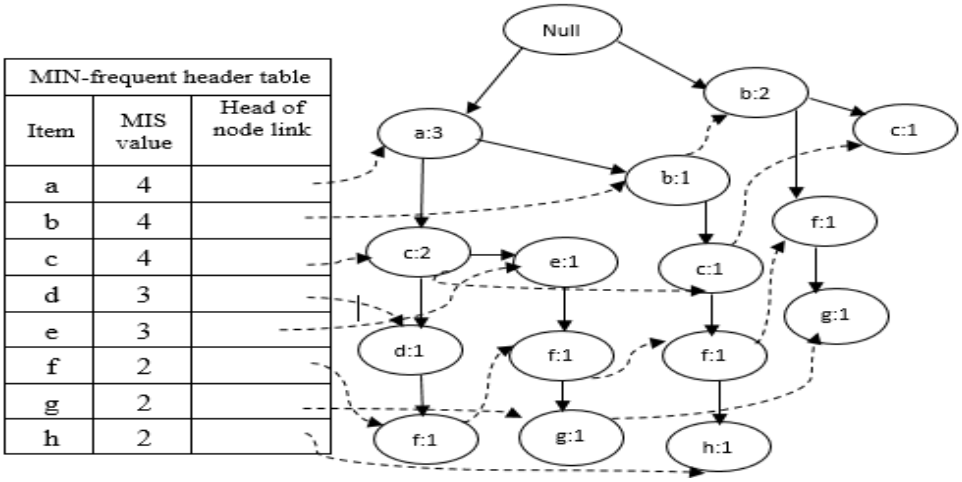

Figure 3.1. MIS-Tree after inserting all transactions [12]

In our example we can notice that node (c: 2) has two child nodes with the same item-name f. Therefore, 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 child nodes (f: 2). After pruning and merging operations, the resulted structure as shown in Figure 3.2 is called the compact MIS-Tree which contains all necessary information needed to extract the whole set of frequent patterns.

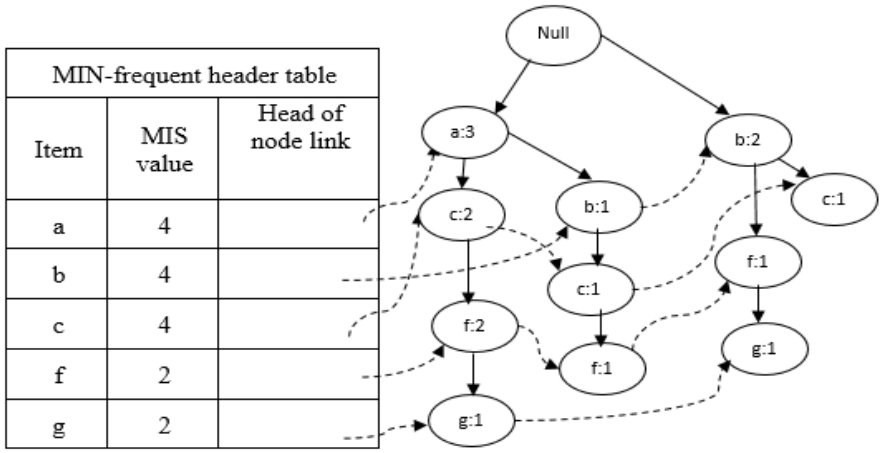

Figure 3.2. The compact MIS-Tree [12]

Mining frequent patterns from MIS-Tree phase

The CFP-growth algorithm works similary to FP-growth algorithm to mine frequent patterns from the compact MIS-Tree. It starts from the bottom of the MIN-frequent-header table to extract the whole set of frequent patterns from the compact MIS-Tree. To explain how this algorithm works, let us illustrate how a conditional pattern base

and conditional MIS-Tree are built for item {f}. Following f's node-link, it occurs in three paths in compact MIS-Tree. The paths are {a:3, c:2, f:2}, {a:3, b:1, c:1, f:1} and {b:2, f:1}. Item f with MIS (f), i.e 2, is a suffix item that has the least MIS among items occur with it since items are ordered in descending order according their MIS. By eliminating item f from these paths, the conditional prefix paths are *{a: 2, c: 2},* {a:1, b:1, c:1} and {b:1}, which called f' conditional pattern base. Notice that because the counter value of item {f} in paths are 2, 1 and 1, the counter of the nodes in these three paths are set to 2, 1 and 1 respectively. Since the item {f} has the least minimum item support threshold among all items that occur with it in these paths, all frequent patterns generated based on the item {f} must have support no less than minimum of suffix item, here item {f} (i.e, 2). Thus, all frequent patterns must have support no less than 2. Otherwise, they cannot be frequent.

After the conditional pattern base of item {f} are identified, the f's conditional MIS-Tree is generated by calculation the counts along the link and searching for patterns that exceed the minimum support threshold value of item {f} (i.e, 2). In the conditional MIS-Tree for suffix item {f}, since only the items {a, b, c }have support no less than the minimum support threshold of item {f} (i.e ,2), the only conditional frequent patterns {(af:2), (cf:2), (bf:2), (acf:2)} are generated. Hence, we generate f's conditional frequent pattern {(af: 2), (cf: 2), (bf: 2), (acf: 2)}. By iterating the same process for the remaining items in MIN-frequent header table, we extract the complete set of frequent patterns as shown in Table 3.6.

Table 3.6. The complete set of frequent patterns using CFP-growth

Item	MIS	Conditional Pattern Base	Conditional MIS-Tree	Frequent patterns
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	-	-	-

There are two performance issues that is found in CFP-growth as follows.

1) Some infrequent items have been considered by CFP-growth to construct compact MIS-Tree where they cannot play any role to generate the frequent pattern.
2) Some infrequent items are examined to build suffix patterns until its respective conditional pattern base is empty by CFP-growth which will never generate any

frequent pattern. Thus, CFP-growth needs additional more time to figure out the whole set of frequent patterns [14].

To tackle these issues, an improved CFP-growth method called CFP-growth++ has been proposed [14]. They used the first frequent item for pruning operation, called the lowest minimum support (abbreviation as, LMS) whereas CFP-growth uses the lowest minimum item support MIN. Furthermore, they proposed a new operation to remove a leaf node that cannot be used to generate any frequent patterns. CFP-growth carries out exhaustive search in constructed MIS-Tree Since *downward closure* property no longer holds in multiple item support framework. In contrary, a *conditional closure* property has been proposed by CFP-growth++, by which this algorithm mines all frequent patterns without carrying out mining process till a conditional pattern base becomes empty. Therefore, execution time and search space can be reduced.

There are also some studies as [13, 15, 17, 18, and 62] that have been proposed to extract frequent patterns with multiple item supports thresholds based on FP-growth method.

In [13], MIS-Tree based on FP-Tree is used to store the crucial information about frequent itemsets. In this research, the main advantage of MIS-Tree is again in avoiding rescanning database since it holds all information that is needed to extract frequent patterns. They showed that using MIS-Tree is significantly better than using Apriori-like structures as MSapriori.

A multi-support method has been proposed in [15], called MS-FP-Growth, which is an extension version of FP-growth method. In this method, instead of using one *minsup* value for all items, the *minsup* value is changed from one level to another. The *minsup* is varied by in increasing and decreasing the *minsup* value for each level (K-itemsets). The *minsup* value is changed as follows.

1. Increasing *minsup* value, the *minsup* value is increased from one level to another.
2. Decreasing *minsup* value, the *minsup* value is decreased for each level.
3. The last one, the *minsup* value is varied randomly (increasing and decreasing).

In case 1 and 2, the variation of *minsup* is done according to the length of the largest itemset in the dataset. Thus, the size of the multiple support (MS) set is equal to the length of the largest itemset in the database. This is done by fixing the first value of the MS set and the remaining values of MS are determined by dividing or multiplying the first value on or by the length of the current itemset. The third case is similarly done except the first values is chosen randomly.

The Central Limit Theorem [21] has been utilized in [17] to identify MIS values of items automatically. An automatic tuning MIS method is used to specify the MIS value for each item by the following equations.

$$\mu(i_j) = \frac{1}{n}\sum_{j=1}^{n} \sup(i_j) \tag{3.4}$$

$$MIS(i_j) = \mu(i_j) - \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\sup(i_j) - \mu(i_j)\right)^2} \tag{3.5}$$

where $\mu(i_j)$ leads to the average frequency of the items $i_1, \ldots, i_n$ that belongs to the same level nodes of each category. In [17], all processes as same as CFP-growth++ except specifying MIS of items as mentioned above.

In [18], they claimed that using only MIS does not represent precisely the occurrence of items in the database. Hence, the characteristics of real world databases, the significance of each item and non-binary values of the items in transactions have to be considered to efficiently reflect the nature of each item. To achieve this, they proposed a novel tree structure, called MHU-Tree (Multiple item supports with High Utility Tree), which is built with a single database scan to maintain information of transactions and high utility of items. Moreover, four pruning conditions were used to reduce the search space and the number of generated itemsets. MHU-Growth (Multiple item supports with High Utility Growth) was proposed also to extract high utility patterns with considering multiple item minimum support thresholds as well as utilities of items.

In [16], another method has been proposed for mining frequent patterns with MIS from incremental database. Since the database is frequently updated, an infrequent item may be frequent or a frequent item may be infrequent. Thus, in this method, two algorithms have been developed for maintenance of the tree after support tuning and incremental update of database without rescanning the database. The MIS-Tree maintenance method that proposed in [12] have been modified by violating the restrictions that were used in it. These restrictions state that all the data items should be same in the MIS-Tree after tuning since it was not possible to recover a deleted data in the MIS-Tree without rescanning the database again. In this method, there is no need to any kind of restrictions since the deleted_node_info table, which is used to store all information about the deleted items, helps to recover deleted node if needed from the MIS-Tree after support tuning by traversing the path according to the prefix and suffix value of data item. Another method has proposed to only scan incremental update of database (d) without the reconstruction of MIS-Tree after update of database.

The above tree-based methods are used to find frequent patterns with MIS by one database scan. Then, several pruning and merging operations are used to reduce the search space and execution time. Although these operations reduce the search space and execution time, they consume time for rebuilding the tree because of infrequent item pruning. They also occupy unnecessary memory space for items that will not be used in association rule generation in the final phase.

CHAPTER 4

MISFP-GROWTH ALGORITHM

The proposed method, Multiple Item Support Frequent Pattern growth (MISFP-growth), is extended version of FP-growth [3]. We utilize FP-growth with a slight difference to mine frequent patterns with multiple support thresholds. The main differences between our method and FP-growth is as follows.

1- FP-growth is used to create frequent itemsets based on single threshold whereas MISFP-growth is used to mine frequent patterns with multiple support thresholds.
2- Items in FP-growth are arranged in descending order in terms of their actual support but in MISFP-growth items are sorted in descending order in terms of their minimum item support threshold values.

MISFP-growth reduces search space based on the minimum of minimum item support threshold, MIN-MIS. This idea plays a big role to reduce search space since it is used to discard unpromising items that play no role in creating frequent patterns at high order. In property 4.1, the minimal minimum item support threshold (MIN-MIS) which plays a big role to reduce search space is defined. Since the items that have support less than MIN-MIS play no role to create any frequent patterns at high order, we use it to discard any item that has support less than MIN-MIS before adding items to MISFP-Tree.

Property 4.1. Discarding property (MIN-MIS). Any item that has support less than MIN-MIS will be discarded and it cannot be used to build MISFP-Tree.

To understand this property, we illustrate the following example.

Example 4.1. Suppose the following items, their actual support and the minimum support of items are given as in Table 4.1.

The least minimum support threshold in this example is the value of item {h} (i.e, 3), any item that have actual support less than 3 will be discarded since no frequent patterns can be generated from it. Thus, items {d, g, h} will be discarded.

Table 4.1. Items, their minimum item support and actual support

Item	a	b	c	d	e	f	g	h
MIS	6	6	6	5	4	3	3	3
Actual support	6	5	5	2	4	3	2	1

Notice, although some items such as {b, c, e} have support less than their minimum item support thresholds, we keep them as they may be used to generate frequent patterns with other items. Assume that we have itemsets {(a, c, f): 3, (b, c, f): 2}, where the number after ":" represents the actual support of these itemsets in a given database. Therefore, the itemset {(a, c, f): 3} is frequent as its support (i.e, 3) is not less than the lowest minimum support threshold (i.e, 3) of items containing in this itemset. Thus, although the item {c} has support 5 that is less than its minimum support threshold 6, it generates frequent itemset with remaining items whose supports are greater or equal than the least minimum item support threshold. The itemset {(b, c, f): 2} is infrequent itemset since its support less than 3.

We build our tree by only those promising items that have support more or equal to MIN-MIS. MISFP-growth utilizes MISFP-Tree, an extended prefix-tree, which compresses all transactions of database in horizontal data format in memory. This enables MISFP-growth to search for the complete set of frequent patterns without the requirement of generating a large number of candidate itemsets. 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, which is used to hold 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 node-link that point to item's occurrences in the MISFP-Tree. Nodes that have the same item-name are linked in sequence. Such node-links simplify tree traversal.
5. Build the conditional pattern base and the conditional MISFP-Tree of each suffix item whose support is greater or equal to its predefined MIS. These two data structures represent the knowledge extracted from MISFP-Tree.

As a summary, MISFP-growth algorithm involves two main steps to mine the whole set of frequent patterns and rare patterns with MIS as follows; construction of MISFP-Tree (steps 1-4) and mining frequent patterns from MISFP-Tree (step 5).

4.1 Construction of MISFP-Tree

We use a new tree structure, called the MISFP-Tree which is an extended version of FP-Tree [3]. It is used to hold all the information that is needed to extract frequent patterns with multiple item support thresholds.

Definition 4.1. Construction of MISFP-Tree. A multiple support frequent pattern tree (MISFP-Tree) is a tree structure that can be defined as follows.

1- It composes of a root named as null, a set of item prefix subtrees as the children of the root, and a MIN-MIS-frequent item header table which contains all items have support more than MIN-MIS.

2- Each node in the item prefix subtree composes of three fields: item-name, count and node link, where item-name represents which item this node presents, count records the number of transactions represented by the portion of the branch reaching this node, and node-link links to the next node in the MISFP-Tree carrying the same item-name, or null if there is none.

3- Each entry in the MIN-MIS-frequent item header table consists of three fields: item-name, item's minimum support thresholds and head of node-link which points to the first node in the MISFP-Tree carrying the item-name.

4- All the items in the table are sorted in descending order in terms of their minimum item support thresholds values.

5- The MISFP-Tree and MIN-MIS-frequent header table consist only of items that have support no less than MIN-MIS.

The process of building MISFP-Tree works as follows;

1- Scan database once to get the actual support of items.

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

3- All items that have support less than MIN-MIS are discarded because they play no role to create any frequent items according to Property 1.

4- Items are sorted in descending order in terms of their multiple item support thresholds.

5- To Build MISFP-Tree, we scan database once again and insert transactions into the tree arranged according their multiple item support thresholds in descending order.

6- After MISFP-Tree is constructed, we extract all frequent patterns from the MISFP -Tree structure which contains the complete information that is needed to create the whole set of frequent patterns.

Input: DB: Database, MIS: minimum item support value for all items in DB.
Output: MISFP-Tree, MIN-frequent header table.
Method:
1: Scan DB once. * count the support of each item
2: Find the MIN-MIS * represents the lowest minimum item support threshold
3: Discard all items that have support less than MIN-MIS
4: Create MIN-MIS-frequent header table * in descending order of MIS
5: Create the root of a MISFP-Tree R, labeled as "null"
6: For each transaction $t \in DB$ do
Sort items in t. * in descending order in terms of their minimum item
* support threshold
Call Insert_MISFP-Tree (p|P, R) * p represents the first item, P represents the remaining
* items and R is root of tree
End for

Figure 4.1. Construction of MISFP-Tree

The steps involving in construction of MISFP-Tree are described in a pseudocode as shown in Figure 4.1.

Line 1 in algorithm 4.1, makes a pass over the database to register the actual support count of each item in database. In line 2, the least minimum support threshold (MIN-MIS) is found. All items that have support less than MIN-MIS are discarded in line 3 whereas the line 5 and 6 are used to insert all items that have support no less than MIN-MIS into MISFP-Tree. Line 4 is used to create MIN-MIS-frequent header table that used to store all distinct items in the tree to facilitate tree traversal.

In procedure shown in Figure 4.2, Line 1 is used to insert all items in specified transaction to the MISFP-Tree. Lines 2 and 3 are used to increment the count of the item by 1 if it has been added before. Lines 5, 6 and 7 are used to create a new node with count equals to 1, to link it to its parent and to link the other nodes carrying the same name via node-link.

```
1:   While (P ≠ null)
2:   if R has a child node M such that p.item-name = M.item-name then
3:        Increase M's count by 1.
4:   else
5:        Create a new node M, and let its count be 1.
6:        Link its parent to R.
7:   Link its node-link to the nodes with the same item-name via the node-link structure.
8:   end if
9:   Call Insert_MISFP-Tree (P, M)
10:  End while
```

Figure 4.2. Insertion into MISFP-Tree

To illustrate how MISFP-growth works, the steps of its process can be best understood using the following example.

Example 4.2. Given a transaction database DB as shown in Table 4.2 and the multiple item supports of items in Table 4.3, construct the MISFP-Tree with MIS in DB.

Table 4.2. Transaction database

TID	Items	Items have support no less than MIN_MIS
1	d , c , a , f	a , c, f
2	g , c , a ,f , e	a, c, f, g
3	b , a , c , f, h	a, b, c, f
4	g , b ,f	b, f, g
5	b , c	b, c

Table 4.3. MIS and actual support of items

Item	a	b	c	d	e	f	g	h
MIS	4	4	4	3	3	2	2	2
Sup	3	3	4	1	1	2	2	1

To build MISFP-Tree from the data in Table 4.2 with the multiple predefined minimum support values in Table 4.3, the process of MISFP-Tree that shown in Figure 4.1 works as follows.

1. Scan database once to find out the support of items in the database DB. The Table 4.3 shows the items, MIS of items and their actual support in the consecutive rows of the table.
2. Find out the least minimum support threshold among all minimum item support thresholds of items: MIN-MIS = min {MIS(a), MIS(b), …, MIS(h)} = 2.
3. Compare the actual support of items with MIN-MIS value (i.e, 2) and each item has support less than 2 is discarded since they play no role in generation of frequent patterns. Hence, items {h, e, d} are discarded. The remaining items that have support no less than 2 are arranged according to their minimum item support thresholds in descending order as shown in the right column in Table 4.2.
4. Scan database once again to construct MISFP-Tree. We use items in the right column of the Table 4.2 to build our tree. The process of inserting transactions into the tree works as follows.
   4.1 The root of MISFP-Tree is created and labeled as "null".
   4.2 For the first transaction {a, c, f}; the first branch of MSFP-Tree is created as shown in Figure 4.3. Notice that all items in the transaction are inserted into the tree in descending order in term of their minimum item support thresholds.
   4.3 For the second transaction {a, c, f, g}; since it shares the prefix {a, c, f} with the first transactions, the count of each node along the prefix is increased by 1, a new node (g: 1) is generated and linked as child of (f:2) as shown in Figure 4.4.
   4.4 For the third transaction { a, b, c, f } , since it shares the prefix {a} in the previous transaction , the count of node {a} will be incremented by 1 , the new path will be created for remaining items , a new node (b:1) is created and linked as child of (a:3), a new node (c:1) also is generated and linked as child of (b: 1) and another new node (f:1) is created as child of node (b:1) as shown in Figure 4.5.
   4.5 By repeating the steps 4.2- 4.4, consecutive transactions are added to the tree. Figure 4.6 shows the complete MISFP-Tree after we insert all transactions.

We generate a MIN-MIS-frequent item header table to facilitate tree traversal. Nodes that have the same item-name are linked in sequence by node-links.

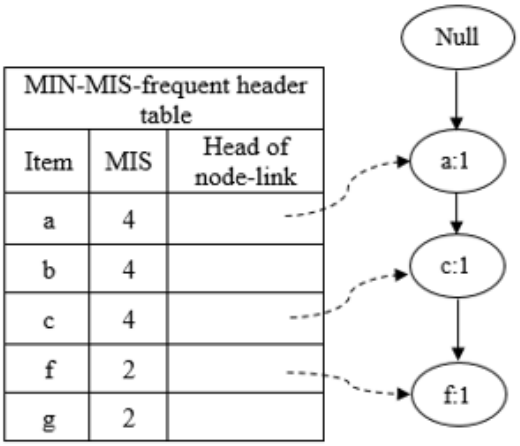

Figure 4.3. After inserting the first transaction

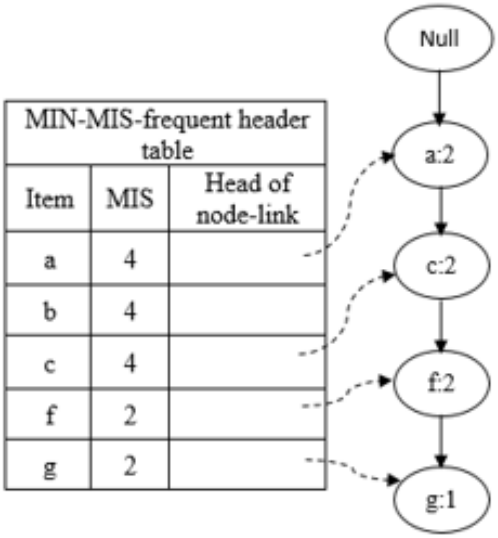

Figure 4.4. After inserting the second transaction

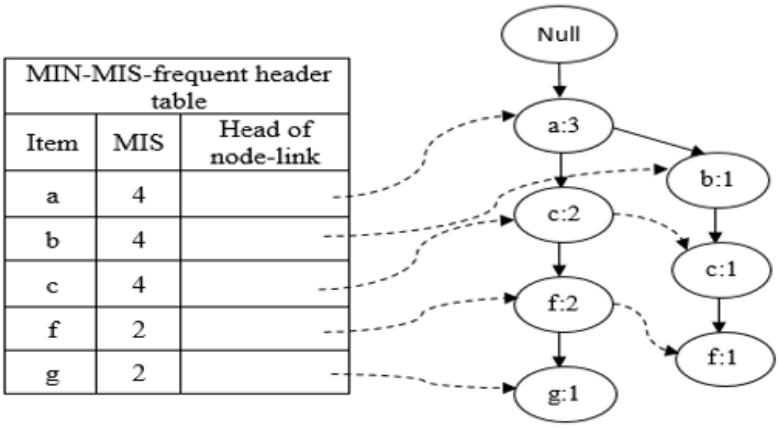

Figure 4.5. After inserting the third transaction

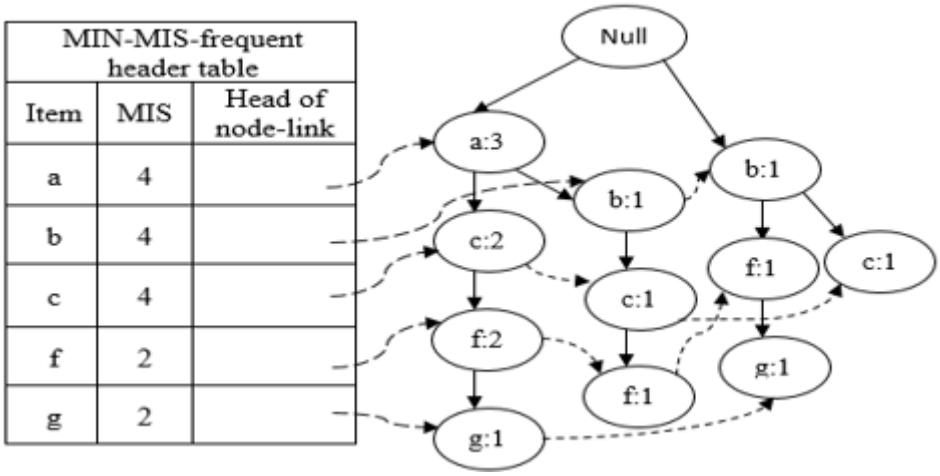

Figure 4.6. MISFP-Tree after adding all transactions

4.2 Mining Frequent Patterns from MISFP-Tree

For the generation of frequent patterns with MIS, the proposed method, MISFP-growth, consists of two main phases. In the first phase, a compact data structure, MISFP-Tree, is constructed which holds crucial information about frequent patterns as we see in the above section. In the second phase, an efficient method is proposed called MISFP-growth algorithm to mine the whole set of frequent patterns from MISFP-Tree. This method is a modified version of FP-growth algorithm [3].

In this method, the process of mining patterns using MISFP-Tree is almost same as mining the FP-tree in FP-growth. The variance between the two methods is as follows. In FP-growth mining process, the conditional pattern base and conditional FP-Tree for

each item are built based on a single *minsup* and then the frequent patterns are created if the support of a pattern is greater than or equals to the *minsup*. However, MISFP-growth method builds the conditional pattern base and conditional MISFP-Tree for each item based on MIS of the suffix item as MIS is specified for each item. Then, a frequent pattern is generated if pattern's support is greater than or equals to predefined MIS of the suffix item.

There are some important definition and properties of MISFP-Tree structure that should be defined.

Definition 4.2 (Conditional pattern base). A pattern P is called x's conditional pattern base if and only if x is in P and MIS(P) = MIS(x).

Example 4.3. Consider three items x, y and z in a given database. Their MIS values are:

MIS(x) = 15%, MIS(y) = 25% and MIS(z) = 30%.

Suppose there is a pattern {x, y, z} with support = 10%.Then, pattern {x, y, z} is an x's conditional pattern since MIS(x) = MIS(x, y, z).

Definition 4.3. (Conditional frequent pattern). A frequent pattern P is called x's conditional frequent pattern if and only if x is in P and MIS(P) = MIS(x).

Example 4.4. Following Example 4.3, assume that the pattern{x, y, z} has support = 20%, then the pattern {x, y, z} is x's conditional frequent pattern since it is frequent and MIS(x) = MIS(x, y, z).

Property 4.2. (Node-link property). For each item x, we can get all possible x's conditional frequent patterns by following x's node-link, starting from x's head in the MIN-MIS-frequent header table.

For example, following f''s node-links in Figure 4.6, we can get all f's conditional frequent patterns from paths : { a, c, f}, {a, b, c, f} and { b, c, f}.

Property 4.3. Downward closure property with MIS. Any itemset contains an item with support less than the lowest minimum support threshold cannot be considered as frequent.

Following the example 4.1, consider the itemset {a, c, g}. Since the actual support of item {g} equals 2, the items {a, c} occur only twice with item {g}. Thus, the support of itemset {a, c, g} can never be more than 2. According to the definition of minimum item support thresholds, MIS of {a, c, g} = min {(MIS (a), MIS(c), MIS (g)} = 3. Therefore, itemset {a, c, g} cannot be frequent since its support always is less than 3.

MISFP-growth is similar to CFP-growth++ [14] in extracting the complete set of frequent patterns from MISFP-Tree with the following difference.

Since we do not discard those items whose support greater than MIN-MIS and less than their predefined MIS, those items can be used to generate frequent patterns with others but from them no frequent pattern can be generated. Therefore, we avoid generating conditional pattern base and conditional MISFP-tree for these items by checking the item's support of each item against its predefined MIS. Then, a frequent pattern is generated if pattern's support is greater than or equals to predefined MIS of the suffix item.

The process of mining frequent patterns from MISFP-Tree works as follows. In order to mine frequent patterns from MISFP-Tree in Figure 4.6, we start to mine frequent patterns that can be created from item {g} since it has the least minimum threshold among all items in the MIN-MIS-frequent table header. Following the node-link of item {g}, there are two branches contain item {g} : { a: 3, c: 2, f: 2, g: 1} and {b:1, f: 1, g: 1}. Considering the item {g} as suffix item, all frequent patterns generated based on item {g} must have support no less than MIS of suffix item, here item {g} (i.e, 2). Since items are ordered in descending order in terms of their minimum item support thresholds, the item {g} has the least minimum thresholds among all items that occur with it in these paths. Therefore, all patterns that have support less than 2 cannot be frequent.

To construct the conditional pattern base and the conditional MISFP-Tree for item {g}, we exclude it from two paths that it occurs with them. Hence, the conditional pattern base of item {g} is { a: 1,c: 1, f: 1} and {b:1, f: 1}. Notice that since the counter value of {g} in each path is 1, the counter of the nodes in these two branches are set to 1. After the conditional pattern base of item {g} are identified, the g's conditional MISFP-Tree is created by adding the counts along the link and searching for patterns that exceed the minimum support threshold value of item {g}.

In the conditional MISFP-Tree for suffix item {g}, since only the item {f }has support no less than the minimum support threshold of item {g} (i.e., 2), the only conditional frequent pattern {(fg:2)} is generated. Since the support count of the remaining items {a, b, c} is 1, no frequent patterns can be created from them. Thus, we create g's conditional frequent pattern (fg: 2). By repeating the same process for the remaining items in MIN-MIS-frequent header table, we find out the whole set of frequent patterns as shown in Table 4.4.

Table 4.4. The complete set of frequent patterns from MISFP-Tree

Suffix Item	*Minsup*	Conditional Pattern Base	Conditional MISFP-Tree	Frequent Patterns
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	-	-	-

CHAPTER 5

PERFORMANCE EVALUATION

In this chapter, the proposed method, MISFP-growth, is compared with the recent tree based method, CFP-growth++ [14], to discover frequent patterns under multiple item support. To verify the effectiveness of and efficiency of the proposed method, several experiments are conducted using five datasets with different characteristics. This chapter is divided into five subsections where in the first, the experimental environment and five datasets are presented. In the subsections, 5.2, 5.3 and 5.4, the experiments that are carried out to measure execution time, memory space and scalability performance of the algorithms respectively are explained. In subsection 5.5, computational complexity of MISFP-growth and CFP-growth++ is analyzed. In the last subsection, discussion of results is explained. Note that, the frequent patterns generated by the MISFP-growth algorithm are the same as those created by CFP-growth++ algorithm in all experiments. This indicates that the experimental results discovered by the proposed method is correct and complete. In the last section, discussion on the performance evaluation results is introduced.

5.1 Experimental Environment and Datasets

We conduct three experiments using different type of datasets to measure the performance of the proposed method, MISFP-growth. All experiments are executed on an Intl(R) core i7 -5500u CPU@ 3.40 GHz with 8GB main memory, running on Microsoft Windows 10 operating system. All the programs are implemented with C#.

We use two kinds of datasets in our experiments; one synthetic dataset (T10I4D100K) and four real world datasets (Kosarak, Pumsb, Retail, and Mushroom). The synthetic dataset T10I4D100K is created with the data generator [2] which is widely used for evaluating association rule mining algorithms. The real world datasets are taken from the FIMI data repository page [51]. The important characteristics of the synthetic and real word datasets are shown in Table 5.1. The experiments are carried on both of sparse (T10I4D100K, Kosarak and Retail) and dense dataset (Pumsb and Mushroom). A

dense dataset is a dataset where each instance (e.g. transaction) are very similar to each other. For example, if you consider a dataset represented by a binary matrix, a dense dataset is the one with a lot of "1"s and few "0"s. In contrast, a sparse dataset is a dataset where transactions are different from each other, in other words, sparse datasets have too many distinct items or/and small average transaction length.

Table 5.1. Characteristics of datasets

Datasets	Density (%) [1]	Size (MB)	# of Distinct Items	Average Transaction Length	# of Transactions
Kosarak	0.002	30.5	41271	8.1	990002
Retail	0.006	4.2	16470	10.3	88126
Pumsb	3.5	16.3	2113	74	49046
Mushroom	19.3	0.56	119	23	8124
T10I4D100k	1,15	3.83	870	10.1	100000

We use the following formula for assigning MIS of items that is based on their actual supports [8]. LS represents the user-specified least minimum item support, $\beta \in [0,1]$ represents the parameter used to control how the minimum support values of items should be related to their occurrence in database and $f(i)$ represents the number of transactions that contain item i (the support of item i).

$$MIS(i) = \begin{cases} f(i) * \beta, & f(i) * \beta > LS \\ LS, & Otherwise \end{cases} \quad (5.1)$$

Notice, if $\beta = 1$ and $f(i) \geq$ LS, then the minimum item support threshold values of items are the actual support of items, $f(i)$, whereas if $\beta = 0$, then there is only one minimum support LS. The same process of mining with single threshold is implemented as FP-growth algorithm. In our experiments the β parameter is calculated by the following formula: $\beta = \frac{1}{\alpha}$

According to this formula, increasing the value of **α** leads to decrease in MIS of items and that increases the number of frequent patterns that are generated. A series of experiments are carried out to measure the performance of MISFP-growth and CFP-growth++ while the value of **α** is increased and value of LS is fixed as we will see in sections 5.2, 5.3 and 5.4**.** For the datasets Kosarak, Retail, T10I4D100k and Mushroom, **α** is varied from 1 to 10 and we set LS at 0.001, 0.001, 0.01 and 0.1 respectively. In dense dataset, Pumsb, **α** is varied from 1 to 1.9 and we set LS at 0.6. Figures 5.1, 5.2, 5.3, 5.4

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

and 5.5 show the number of frequent patterns, which are generated with respect to **α.** It can be seen from the graphs that increasing **α** leads to increase in the number of frequent patterns for all datasets. This is due to the fact that at higher values of α, MIS' values of items are decreased and they become closer to LS threshold.

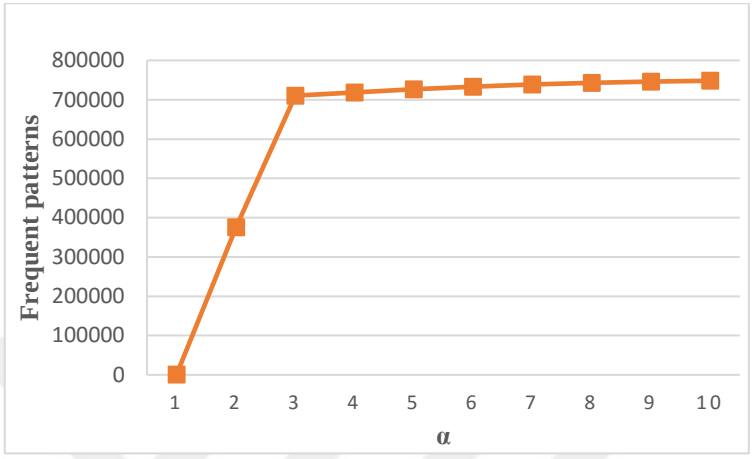

Figure 5.1. Frequent patterns in Kosarak dataset

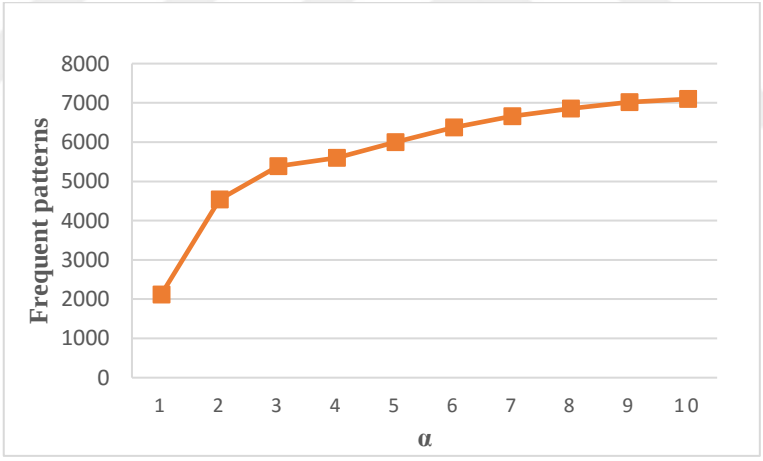

Figure 5.2. Frequent patterns in Retail dataset

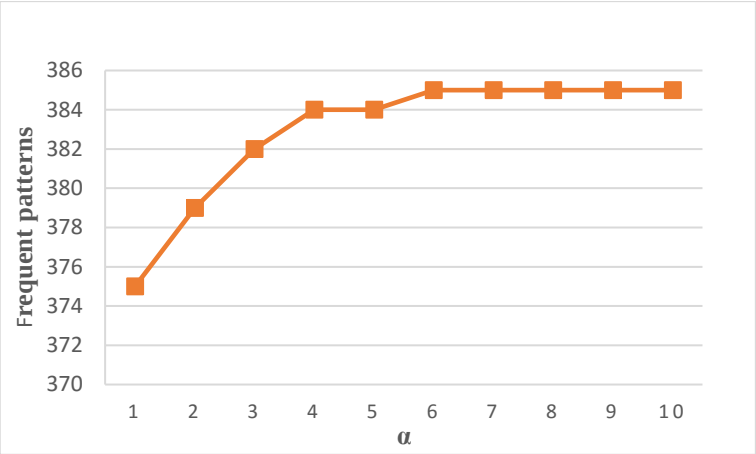

Figure 5.3. Frequent patterns in T10I4D100K dataset

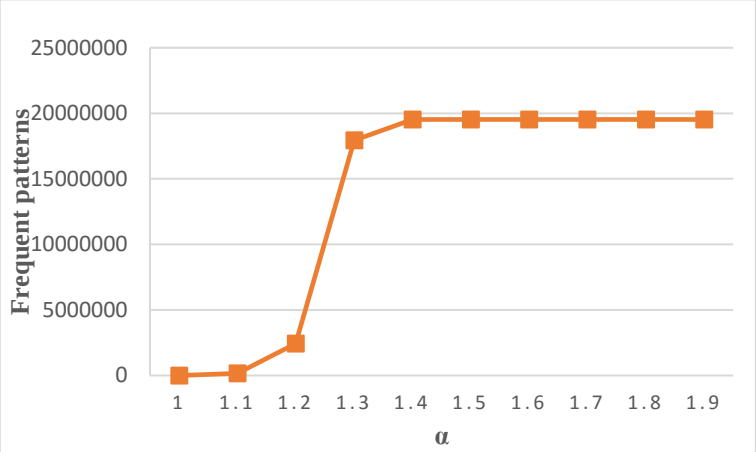

Figure 5.4. Frequent patterns in Pumsb dataset

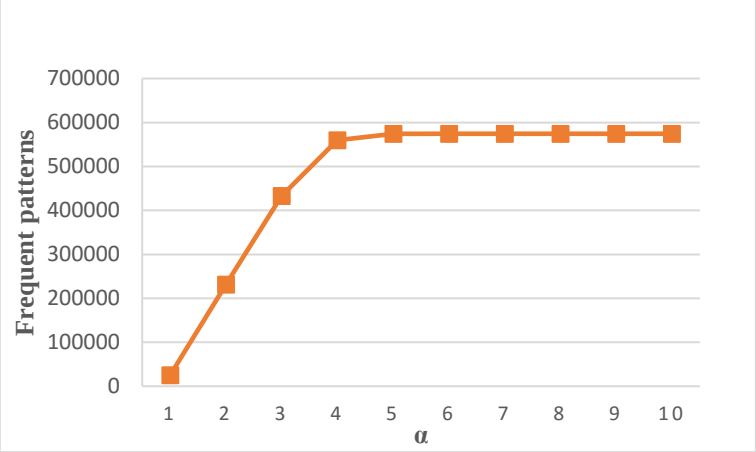

Figure 5.5. Frequent patterns in Mushroom dataset

5.2 Execution Time

The execution time performance of our proposed method, MISFP-growth and the recent tree-based algorithm, CFP-growth++ is measured on four real world datasets (Kosarak, Pumsb, Retail, and Mushroom) and one Synthetic dataset (T10I4D100K). In this experiment, the values of **α** is changed and LS is fixed.

For the real world sparse dataset, Kosarak, we fix the LS = 0.001 and **α** is changed from 1 to 10. In the Figure 5.6, it can be clearly seen that the performance of the proposed method, MISFP-growth, is significantly better than CFP-growth++ performance in terms of execution time. Kosarak dataset is so sparse, lots of items that are used to build MIS-Tree have to be discarded and CFP-growth++ rebuilds the compact MIS-Tree. Therefore, the execution time that is required by MISFP-growth is significantly less since rebuilding phase is not required.

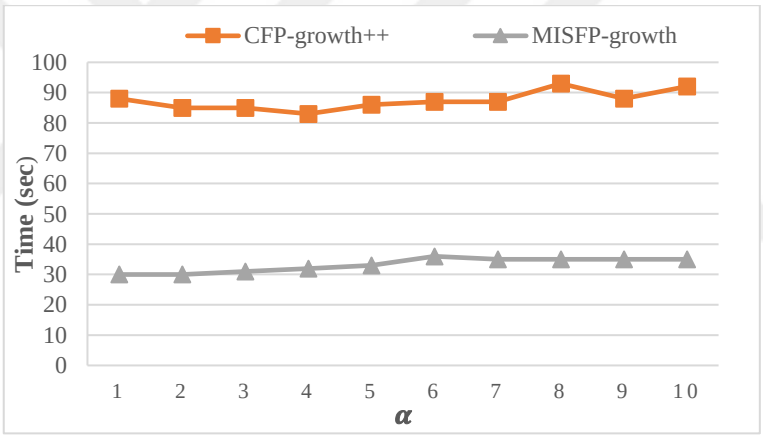

Figure 5.6. Execution time for Kosarak dataset

For the sparse datasets Retail, we set LS = 0.001 and the parameter **α** is changed from 1 to 10. Figure 5.7 shows the performance of proposed method at various **α** values for real world dataset Retail. It can be seen that the execution time that is required by proposed method is significantly less than CFP-growth++. This is due to the effect of the time that is required by CFP-growth++ to re-build the MIS-Tree since there are lots of infrequent items that have to be discarded. In the proposed MISFP-Tree, there is no need to discard any items since it is constructed by only those items that have support no less than MIN-MIS.

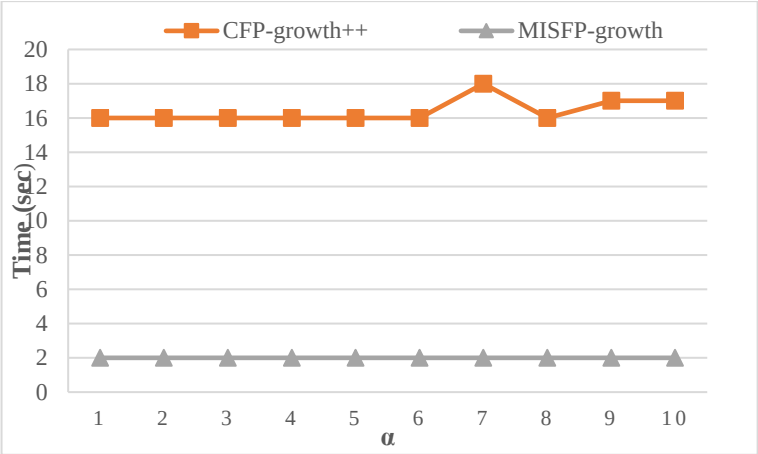

Figure 5.7. Execution time for Retail dataset

In the experiment on the Pumsb dataset, we fix the value of LS = 0.6 and the parameter **α** is changed from 1 to 1.9. Note that, this real dense dataset has a large number of distinct items and the average of transaction is large as well. For example, when the value of **α** is set to 1.5 and LS = 0.6, the number of frequent itemsets discovered from the Pumsb dataset is about 2 millions. Figure 5.8 shows the execution time of proposed method and CFP-growth++. In this graph, we can notice that the proposed method, MISFP-growth, is much faster than CFP-growth++. This is mainly due to that, the MISFP-growth decreases execution time effectively by building an efficient tree, MISFP-Tree without any unpromising items.

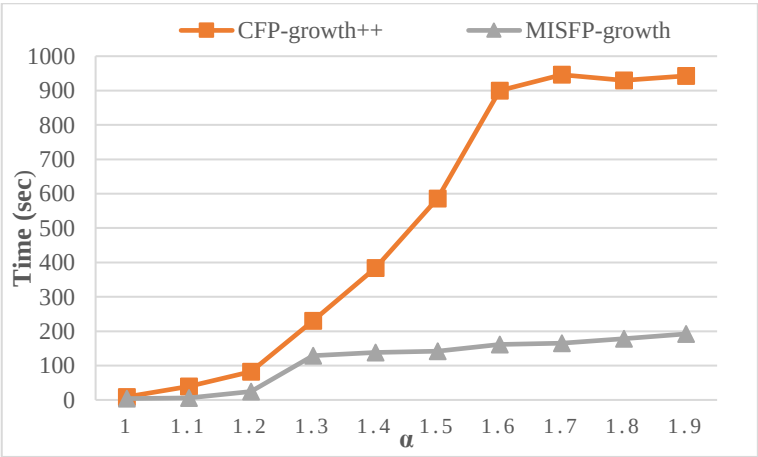

Figure 5.8. Execution time for Pumsb dataset

For the synthetic dataset, T10I4D100K**,** we set LS to 0.01, the parameter **α** is changed from 1 to 10. Figure 5.9 shows the execution time consumed by MISFP-growth comparing with CFP-growth++. It is obvious from the graph that the proposed algorithm,

MISFP-growth, consumes less time. This is due to the fact the lots of items can be discarded from MIS-Tree by CFP-growth++. Therefore, MISFP-growth performance is better than CFP-growth++ performance for the synthetic dataset T10I4D100K in terms of the execution time.

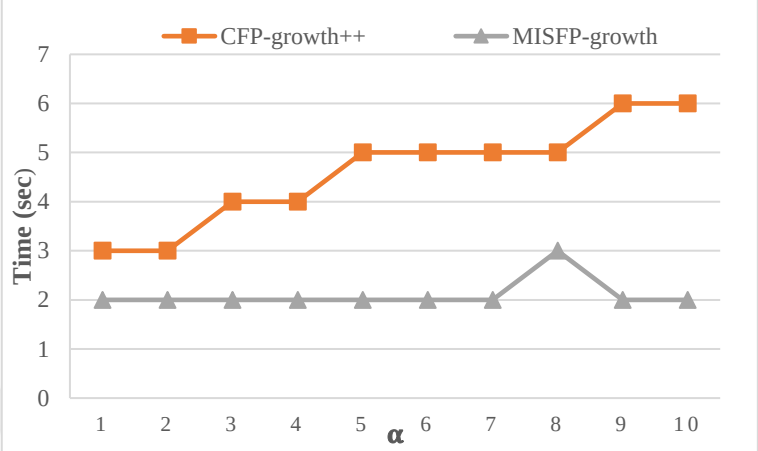

Figure 5.9. Execution time for T10I4D100K dataset

For the real world dense dataset, Mushroom, the LS is set at 0.1 and **α** changed from 1 to 10 as shown in the Figure 5.10. From the experimental result on this graph, it can be noticed that the execution times are approximately the same. Although the performance seems almost same, the performance of the proposed approach, MISFP-growth, is still faster.

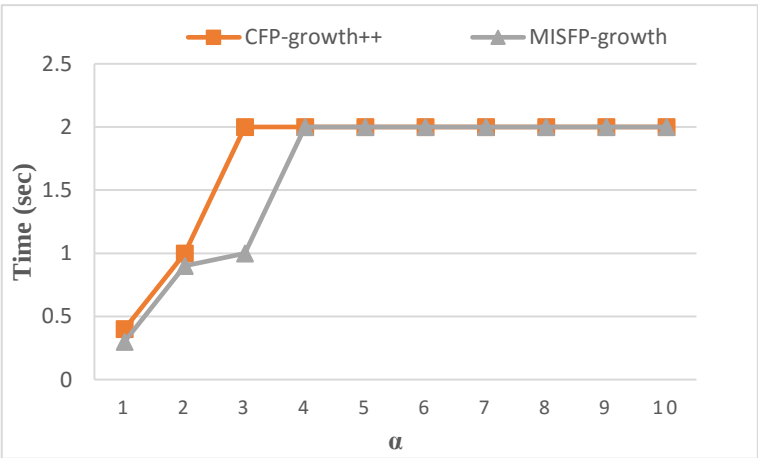

Figure 5.10. Execution time for Mushroom dataset

The reason why overall performance of the proposed method and the compared algorithm, CFP-growth++, is almost same as follows; in dense dataset, a few items have

to be discarded since most of them are frequent. So, there is no need to discard lots of items in MIS-Tree.

The speed-up of the proposed method is summarized in Table 5.2. In this table, Column 4 contains the minimum speed-up (MIN) and the maximum speed-up (MAX) of MISFP-growth against the compared method. The speed-up is defined as the ratio between the execution time of CFP-growth++ and MISFP-growth. Speed-up can be up to magnitude of 8-9 for sparse dataset like Retail. On the other hand, on a dense dataset like Mushroom, execution time of MISFP-growth can be half of CFP-growth++. As a summary, from this table it can be seen that MISFP-growth performs much better than CFP-growth++ for all sparse and dense datasets.

Table 5.2. The speed-up of MISFP-growth on five datasets

Datasets	Density	Varied (**α**)	Speed-up[2] [MIN, MAX]
Kosarak	0.002	[1, 2, …, 10]	[2.4, 2.9]
Retail	0.006	[1, 2, …, 10]	[8, 9]
Pumsb	3.5	[1.1, 1.2, …, 1.9]	[1.8, 6.5]
T10I4D100k	1.15	[1, 2, …, 10]	[1.5, 3]
Mushroom	19.3	[1, 2, …, 10]	[1, 2]

5.3 Memory Usage

Experiment is carried on to compare memory usage performance of MISFP-growth and CFP-growth++ on the datasets given in Table 5.1. Similar to execution time experiment, we change **α** and fix the value of LS. Figures 5.11 - 5.15 show memory consumption of the two algorithms on these datasets.

[2] Speed-up = execution time of CFP-growth++ / execution time of MISFP-growth

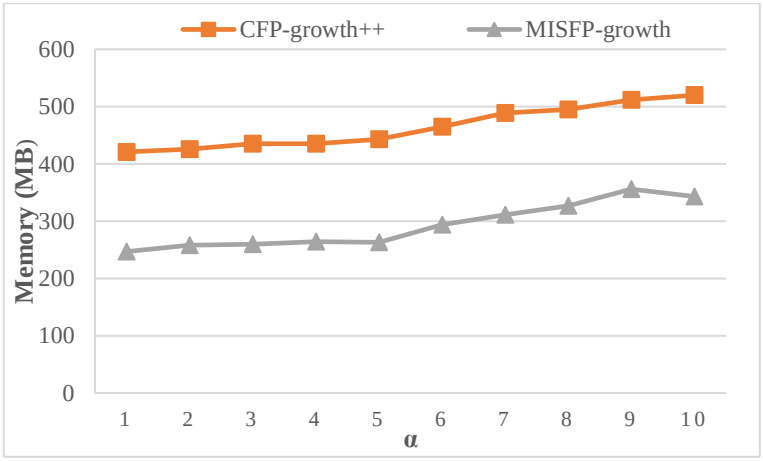

Figure 5.11. Memory usage for Kosarak dataset

For the real world sparse dataset, Kosarak, we fix LS = 0.001 and **α is** changed from 1 to 10. Figure 5.11 shows the memory space, which is consumed by MISFP-growth and CFP-growth++. It can be observed from the graph that the memory space increases as **α** increases**.** This is due to the fact that increasing α leads to decrease the MIS of items. It can be seen that at all **α** values, MISFP-growth performance is significantly better than the CFP-growth++.

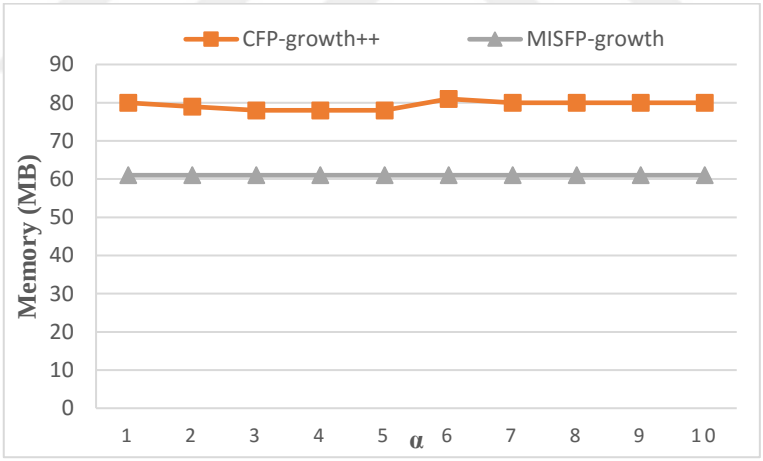

Figure 5.12. Memory usage for Retail dataset

For real world sparse dataset, Retail, the parameter **α** is changed from 1 to 10 and we set LS as 0.001. It can be noticed from the Figure 5.12 that the memory usage appears to be almost linear for both of MISFP-growth and CFP-growth++. This is due to the small difference between the number of frequent patterns generated while increasing the value of parameter **α**. It can be seen from graph that at all **α** values, MISFP-growth consumes less memory than CFP-growth++.

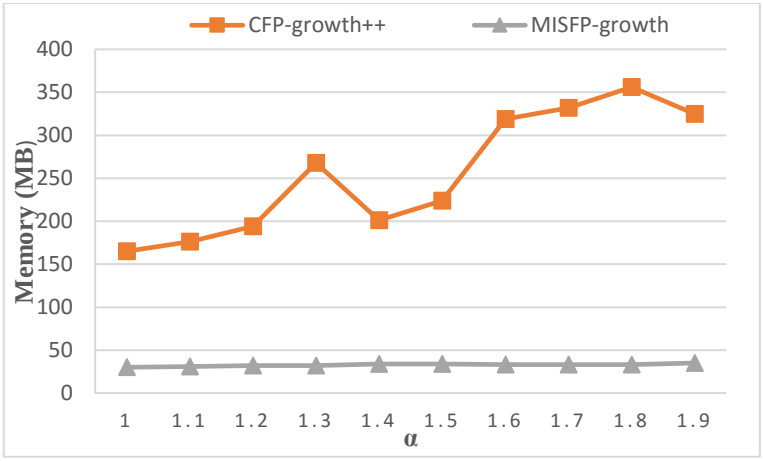

Figure 5.13. Memory usage for Pumsb dataset

For the real world dense dataset, Pumsb, memory consumption of the proposed approach and CFP-growth++ is shown in the Figure 5.13 under varied **α**. The **α** values are changed from 1 to 1.9 and we fix LS = 0.6. It can be observed from the graph that at all **α** values, CFP-growth++ consumes a huge amount of memory whereas MISFP-growth consumes less memory. Although the number of frequent patterns generated becomes larger as the number of **α** increases as shown in Figure 5.4, MISFP-growth consumes less memory than CFP-growth++. This is due to the effect of larger size MIS-Tree of CFP-growth++ algorithm since it contains unnecessary items.

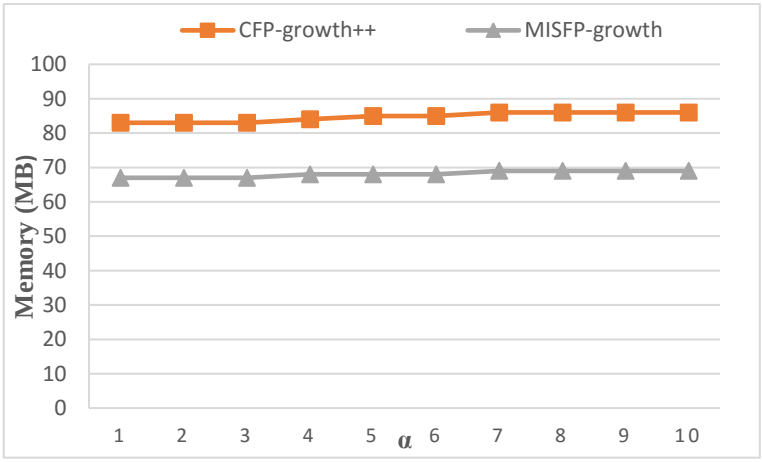

Figure 5.14. Memory usage for T10I4D100K dataset

For the synthetic dataset, T10I4D100K**,** Figure 5.14 shows the performance of the two algorithms as the values of **α** increases from 1 to 10 and LS = 0.01. It can be noticed from the graph that the memory consumption of the two algorithms appears to be linear. This is also because of the number of frequent patterns, which are generated with MIS

while increasing the value of **α**, are approximately constant as shown in Figure 5.3. As it can be observed, MISFP-growth is still better than CFP-growth++ in terms of memory consumption on the synthetic dataset, T10I4D100K.

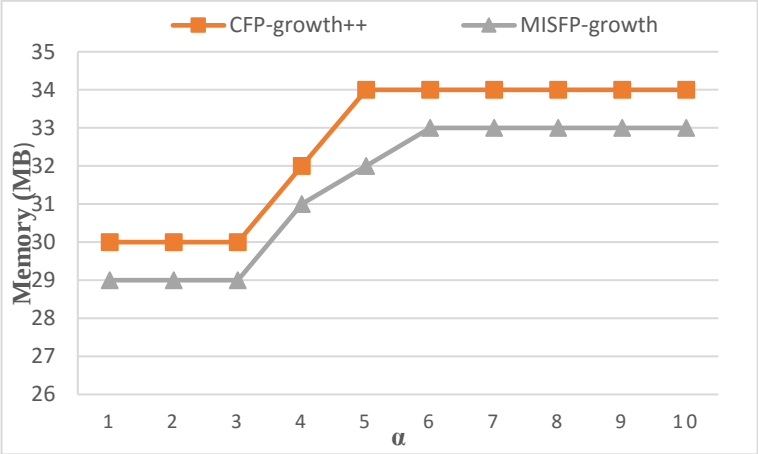

Figure 5.15. Memory usage for Mushroom dataset.

For the real world dense dataset, Mushroom, memory consumption of MISFP-growth and CFP-growth++ is shown in the Figures 5.15 under varied **α**. The **α** values are changed from 1 to 10 and LS = 0.1. It can be observed from the graph that the memory usage of both algorithms increases with increasing **α**. It can be seen from the graph that at all **α** values, the difference in the memory consumption of MISFP-growth and CFP-growth++ is small. This is because a few items have to be discarded in very dense datasets.

Table 5.3 shows the memory gain of the proposed method compared to CFP-growth++ under varied **α**. Column 4 in this table shows the minimum memory gain (MIN) and the maximum memory gain (MAX) of MISFP-growth against the compared method. Memory gain is defined as the percent of the difference between the peak memory consumed CFP-growth++ and MISFP-growth divided by the peak memory consumed by CFP-growth++. It can be noticed that MISFP-growth consumes less memory in all cases except in Mushroom dataset where the memory consumption of the proposed method is slightly less than memory consumption of the compared method. For a quite dense dataset like Pumsb, memory gain can reach up to 90%, on the other hand for a very dense dataset like Mushroom, memory gain can be 6% only.

Table 5.3. Memory gain of the proposed method on five datasets

Datasets	Density	Varied (α)	Memory Gain[3] (%) [MIN, MAX]
Kosarak	0.002	[1, 2, …, 10]	[30, 41]
Retail	0.006	[1, 2, …, 10]	[22, 25]
Pumsb	3.5	[1.1, 1.2, …, 1.9]	[81, 90]
T10I4D100k	1.15	[1, 2, …, 10]	[19, 20]
Mushroom	19.3	[1, 2, …, 10]	[3, 6]

5.4 Scalability

In this section, we compare the performance of MISFP-growth and CFP-growth++ in terms of the size of datasets. Several measurements are carried on to find out how MISFP-growth scales up as the number of transactions increases in the datasets. In fact the aim of these measurements is to show how the proposed data structure scales up with the increasing number of transactions. To verify the performance of the two algorithms, four real and one synthetic datasets, which are shown in Table 5.1, are used in the evaluation of mining frequent patterns under MIS while varying the size of datasets. The datasets are decomposed into 10 even sections with approximately 10% of the dataset size. We measure the execution time for each part after aggregating it with previous parts. We set the parameter **α** to 4 as it is stated in [8] that this value is common in many real world applications.

For Kosarak dataset, we set LS = 0.001. Figure 5.16 shows the performance of two algorithms with respect to the size of Kosarak dataset. It can be observed from the graph that the execution time increases as the size of dataset increases. From the graph, we can see that MISFP-growth scales better than CFP-growth++ as the number of transactions is increased.

[3] Memory gain = ((Memory consumption of CFP-growth++ - Memory consumption of MISFP-growth) / Memory consumption of CFP-growth++) * 100

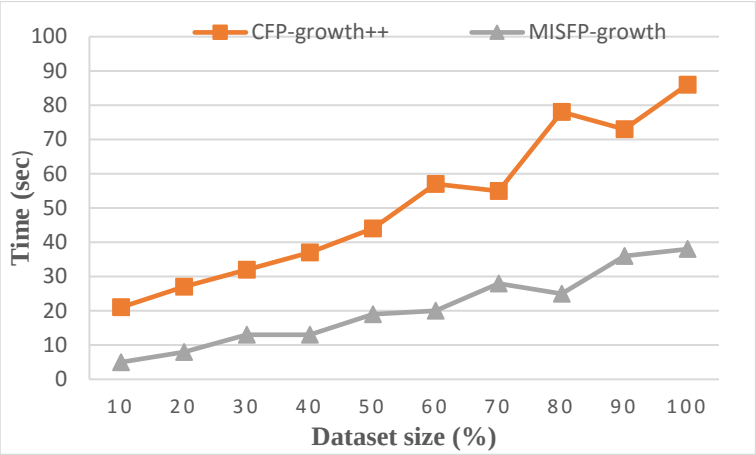

Figure 5.16. Scalability for Kosarak dataset

For the Retail dataset, the parameter LS is set at 0.001. Figure 5.17 illustrates the effect of dataset size on the performance of our proposed method, MISFP-growth, and CFP-growth++. As it can be noticed, MISFP-growth scales much better than CFP-growth++. This is because our proposed data structure is efficiently constructed without any useless items.

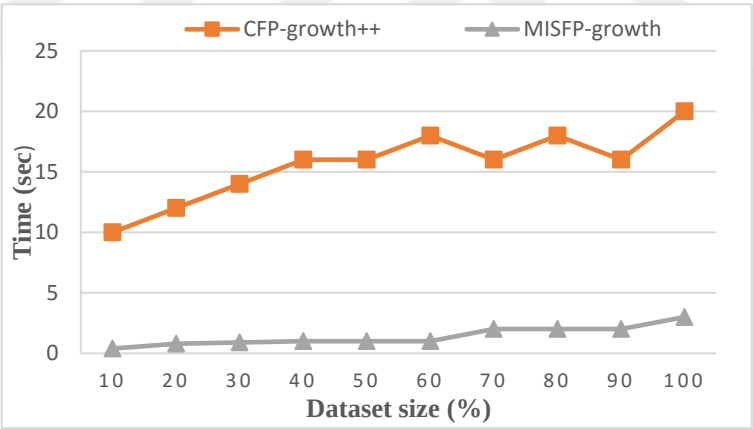

Figure 5.17. Scalability for Retail dataset

Similar results are also obtained for Pumsb dataset with LS = 0.7 as shown in the Figure 5.18. Figure shows that the execution time sharply increases with the increasing dataset size in CFP-growth++, while MISFP-growth seems unaffected. We can see that with Pumsb dataset, MISFP-growth is substantially much better than CFP-growth++ as the size of dataset increases.

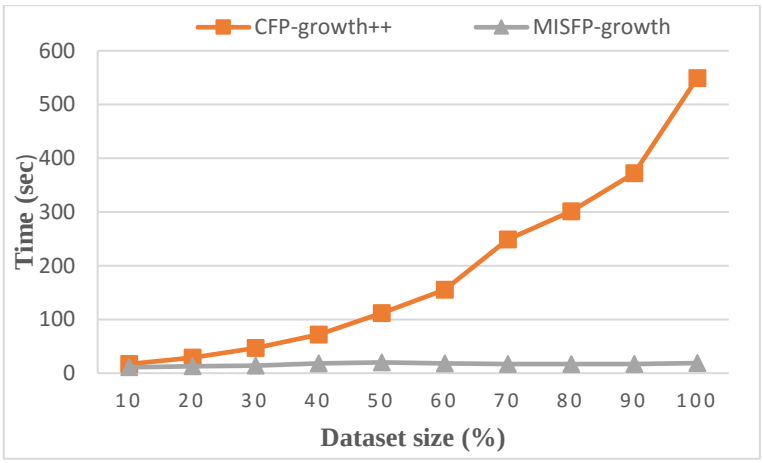

Figure 5.18. Scalability for Pumsb dataset

For the synthetic dataset, T10I4D100K, the parameter LS is set to 0.01. Similar to real world datasets, our proposed method is much better than CFP-growth++ under varied dataset size as shown in Figure 5.19 in terms of scalability.

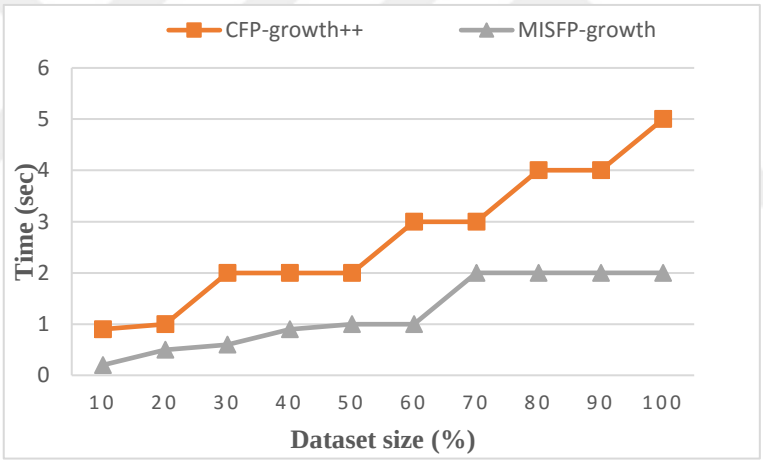

Figure 5.19. Scalability for T10I4D100K dataset.

For the very dense dataset, Mushroom, Figure 5.20 illustrates the performance of two methods under varied datasets size with LS = 0.1. As it can be seen, the execution time decreases with the increasing dataset size in the two methods. This is due to the effects of the decreased paths, which is created while constructing the tree in both methods. The performance of two methods is almost same since most of items in dense datasets are frequent and there is no need for pruning and merging operation in MIS-Tree. It can be observed from the graph that our method is slightly better than CFP-growth++.

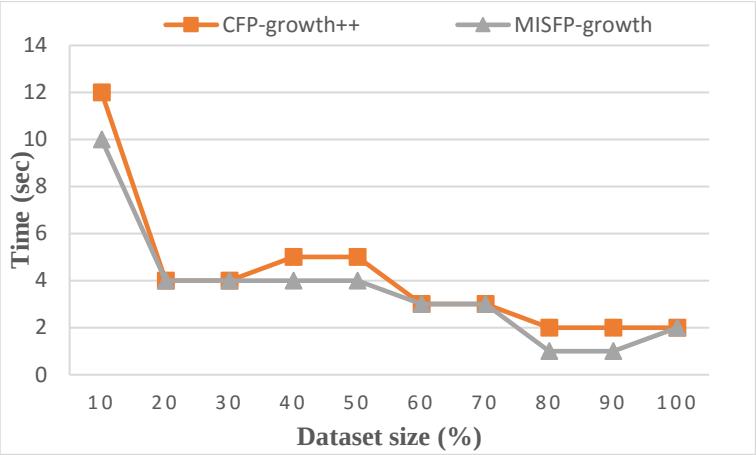

Figure 5.20. Scalability for Mushroom dataset

Table 5.4 shows the speed-up and the overall performance of the proposed method on all datasets under varied dataset size. Column 4 in this table stands for the minimum speed-up (MIN) and the maximum speed-up (MAX) of MISFP-growth against the compared method. As it can be observed from the table that MISFP-growth speeds up much better than CFP-growth++ on sparse datasets; it can be up to magnitude of 28.9. On the other hand we see relatively less speed-up between the ranges of 1 to 2 for dense datasets.

Table 5.4. The speed-up of MISFP-growth on five datasets

Datasets	Density	Dataset size (%)	Speed-up[4] [MIN, MAX]
Kosarak	0.002	[10, 20, …, 100]	[2, 4.2]
Retail	0.006	[10, 20, …, 100]	[6.7, 18]
Pumsb	3.5	[10, 20, …, 100]	[1.5, 28.9]
T10I4D100k	1.15	[10, 20, …, 100]	[1.5, 4.5]
Mushroom	19.3	[10, 20, …, 100]	[1, 2]

5.5 Computational complexity

In this subsection, computational complexity of MISFP-growth and CFP-growth++ is analyzed. In complexity analysis we only care about what happens to the instruction-counting function as the algorithm input (n) grows large. Therefore, we drop

[4] Speed-up = execution time of CFP-growth++ / execution time of MISFP-growth

all the terms that grow slowly such as (assigning value to a variable, comparing two values, etc.) and only keep the ones that grow fast as n becomes larger like (loops, searching for an item in an array, etc).

5.5.1 Computational complexity of MISFP-Tree

Computational cost of building MISFP-Tree is bounded by the number of transactions in a dataset. MISFP-Tree involves the following main steps to build the tree as described in a pseudocode in Figure 4.1:

- Scanning database once to find the support of items. This involves O(|DB|) time, where |DB| is the number of transactions**.**
- In line 2, a loop has to run to find MIN-MIS. This requires O(|MIS|), where MIS stands for the number of MISs.
- Passing over the database once again to add all transactions to the tree. The order cost is O(|DB|) time**.**
- An inner loop iteration runs to add the items of a transaction to MISFP-Tree in Figure 4.2. This order is O (average transaction length).

The computational complexity of building MISFP-Tree is as follows:

T(MISFP-Tree) = O(|MIS|) + O (|DB|) + O (|DB|) * average transaction length). (5.2)

5.5.2 Computational complexity of MIS-Tree

The computational complexity of building initial MIS-Tree is similar to MISFP-Tree. Thus, analysis of a pseudocode of the initial MIS-Tree as shown in Figure 5.21 would be as follwos.

- A loop in line 4 iterates to add all transactions to the MIS-Tree. This needs O(|DB|) time.
- For each transaction, an inner loop repeats in line 1 in the called procedure, Insert_MIS-Tree, in Figure 5.22 to add the items of this transaction to the tree. This order is O(average transaction length).
- A loop repeats to find LMS in line 9. This operation costs O(n), where n stands for the number of MISs.

Thus, this order is O ((|DB|) * average transaction length) which is similar to building MISFP-Tree.

```
Input: DB, MIS
Output: Initial MIS-Tree, LMS
1:  Insert items into the MIS-list with S = 0
2:  Create the root of a MIS-Tree, Tree, and label it as “null”.
3:  For each transaction t ∈ DB do
4:      Sort all the items in t in L order.
5:      Count the support values of any item i, denoted as S(i) in t.
6:      Let the sorted items in t be [p|P], where p is the first element and P is the remaining list.
7:      Call Insert_MIS-Tree([p|P],T).
8:  End for
9: Find LMS  \*  search for the lowest minimum support.
```

Figure 5.21. Construction of initial MIS-Tree [14]

```
Insert_MIS-Tree([p|P],T)
1:  While (P ≠ null)
2:  If T has a child node N such that p.item-name = N.item-name then
3:      Increment N’s count by 1.
4:  Else
5:      Create a new node N, and let its count be 1.
6:      Let its parent link be linked to T.
7:      Let its node-link be linked to the nodes with the same item-name via the node-link structure.
8:  End if
9:  Call Insert_MIS-Tree([P,N).
10:  End while
```

Figure 5.22. Insertion into MIS-Tree[14]

5.5.3 Computational complexity of compact MIS-Tree

The cost of the additional approaches which involving in the CFP-growth++ to reconstruct MIS-Tree is analyzed as follows.

Pruning unpromising items:

The complexity cost of pseudocode of pruning approach in Figure 5.23 is analyzed. The computational cost for this process depends on the following.

- A loop iterates to search for each useless item, m, of the header table in line 1. This order is O(m) time.
- An inner loop repeats to traverse through paths in MIS-Tree for each useless item, m, in the header table. This depends on the depth of the tree by following the node-link of the item m as it can be seen in line 4 in Figure 5.23. Maximum depth of the tree is upper-bounded by the number of transactions in a dataset, N. Thus, this operation requires O(N) time.

$$T(\text{Pruning items}) = O(m*N), \tag{5.3}$$

where m stands for number of items in the header table and N represents the maximum depth of the tree.

Merging child nodes:

To merge the child nodes that carry the same name and linked to the same parent node, pairs of child nodes are merged to create a new node with support equals to the summation support of merged nodes. Analyzing the computational cost of pseudocode of merging approach in Figure 5.24 works as follows.

- For each item in the header table, N, a loop iterates in line 1 in Figure 5.24, to search for any node has tow child nodes carry the same name to be merged. This iteration costs O(N) time.
- An inner loop also runs in line 4 to search for nodes that have same name, N, via the node-link of N in the tree. In the worst-case, this approach requires O(N) time.

The computational complexity of merging approach depends on searching of paths in MIS-Tree for each item of the header table. This also depends on the depth of the tree, N. Thus, the order is O(number of items in header table * maximum depth of MIS-Tree) = O(N*N). In the worst-case scenario, the number of paths that are traversed, for each item in the header table, equals to number of items in the header table and each path

in the tree has a unique transaction. In this case, the computational cost of merging approach requires $O(N^2)$, where N represents the number of transactions.

Thus, computational cost of merging approach in CFP-growth++ is $O(N^2)$ time which is very high complexity that consumes a lot of time.

$$T(\text{compact MIS-Tree}) = O(m*N) + O(N^2) = O(N^2). \quad (5.4)$$

As a result, construction of MISFP-Tree in MISFP-growth runs in linear time as its time execution is directly proportional to the size of dataset whereas a compact MIS-Tree in CFP-growth++ runs in quadratic time since its time execution is proportional to the square of the size of dataset.

```
MISpruning(Tree T, Header table m, LMS)
1: While (mi ≠ null)            \* for each item in the header table
2:    If (S[mi] < LMS) then      \* if the support of an item mi less than LMS
3:      Delete the item mi in header table.
4:      For each node in the node-link of mi in T do
5:        If the node is a leaf then
6:          Remove the node directly.
7:        Else
8:          Remove the node and then its parent node will be linked to its child node(s).
9:        End if
10:     End for
11:   End if
12: End while
```

Figure 5.23. MIS pruning operation [14]

```
MISMerging(Tree T)
1: For each item n in the HeaderTable do
2:   For each node in the node-link of n in T do
3:      If there are child nodes with the same item-name then then
3:        Merge the nodes and set the count as the summation of these nodes' counts.
4:      End if
5:   End for
6: End for
```

Figure 5.24. MIS merging operation[14]

Therefore, computational cost of MISFP-growth and CFP-growth++ can be affected by the following factors;

1- **The lowest support threshold (LS)**:

Decreasing LS often leads to increase the number of frequent patterns. This results in an adverse effect on the computational complexity of CFP-growth++ and MISFP-growth because a bigger tree must be generated with lower LS. Therefore, the algorithms need to make more traversals through the tree to create frequent patterns. On the other hand, increasing LS leads to decrease the number of frequent patterns. Although the frequent patterns decreased which result in decreasing the computation cost in MISFP-growth as it is expected, the computational time is increased in CFP-growth++. This is because CFP-growth++ requires an exhaustive search through the tree to employ pruning and merging operations that are used to reconstruct the tree. Thus, its time cost rises with increasing LS.

2- Number of items:

More computational time will be needed in case the number of useless items grows large in CFP-growth++. This is due to the additional time that is required to prune those unpromising items from the tree. In contrast, MISFP-growth does not consume any time to rebuild the tree as it is built with promising items.

3- The size of dataset:

Increasing the size of a dataset has an adverse effect on the computational cost for both of MISFP-growth and CFP-growth++. If the number of useless items increases, more computation time will be needed in CFP-growth++ as the number of MIS-Tree traversals that are performed during reconstructing MIS-Tree is increased.

5.6 Discussion on the Performance Evaluation Results

We carry out tests with our proposed method and CFP-growth++ and compare time, memory usage and scalability performances under varied **α** and the size of datasets**.** Tests are carried on both real world and synthetic datasets with different characteristics as density, size, distinct items, average transaction length and number of transactions.

For the execution time, the performance of two algorithms on five datasets with various **α** is measured. The experimental results reveal that MISFP-growth is substantially faster than CFP-growth++ for all datasets except in the very dense dataset, Mushroom, where the performance of two methods is almost same. For the sparse datasets, Kosarak, Retail and T10I4D100K, MISFP-growth achieves speed-up from 2.9 to 9 times over CFP-growth++. For the quite dense dataset, Pumsb, MISFP-growth provides speed-up of 6.5. In very dense dataset, Mushroom, MISFP-growth achieves speed-up between 1 and 2. This is due to the fact that there is few items that have to be discarded in this dataset during rebuilding MIS-Tree in very dense datasets. In general, MISFP-growth is about 3 to 9 orders of magnitude faster than CFP-growth++. MISFP-growth effectively decreases consumption of time since it trims the search space by utilizing only the useful items that have supports greater than or equal to MIN-MIS during construction of MISFP-Tree. Furthermore, the proposed method avoids pruning and merging techniques which often greatly reduces the cost of tree rebuilding. In addition, in the mining process we avoid generating patterns from unpromising items by skipping building conditional pattern base and conditional MISFP-Tree for items whose support is not greater than or equals its predefined MIS. In contrast, the MIS-Tree is constructed with all items then pruning and merging techniques are used to rebuild the tree.

We evaluate memory usage of both algorithms on the same datasets as well. MISFP-growth consumes much less memory than CFP-growth++ for all five datasets. We can observe that the memory consumption of MISFP-growth is much less than CFP-growth++ on both sparse and dense datasets. For Kosarak, retail, Pumsb, T10I4D100k and Mushroom datasets, MISFP-growth respectively consumes 41%, 25%, 90%, 20% and 6% times less memory than CFP-growth++. MISFP-growth is so memory efficient since it uses only promising items from a database to construct the tree, which does not requires to maintain much information in memory for useless items. However, CFP-growth++ builds both of tree and the header table by all items in the database, which requires additional memory.

Finally, a series of tests are also conducted to reveal how the two methods scale up as the size of the datasets increases. The measurements are done on the same five datasets. MISFP-growth algorithm scales much better than the CFP-growth++ algorithm with respect to increasing the size of datasets. For the sparse datasets, MISFP-growth achieves speed-up of 4.2, 18 and 4.5 times faster than CFP-growth++ for Kosarak, Retail and T10I4D100k, respectively. For quite dense dataset, Pumsb, MISFP-growth provides

speed-up of 28.9 over CFP-growth++. In very dense dataset, Mushroom, MISFP-growth provides speed-up of 2. This is due to the effect of increasing the number of useless items that CFP-growth++ must process by applying pruning and merging operations. Discarding infrequent items by searching through the tree becomes very expensive in MIS-Tree. It can also be seen that the performance of two methods while the size of transactions increase is almost same in very dense dataset, Mushroom, as few useless items are discarded and most of items are frequent while mining with MIS. In general, the experimental results demonstrate significant scale up by MISFP-growth and its performance continues to improve with increasing size of datasets.

As a result, experimental results show that MISFP-growth significantly outperforms CFP-growth++ on both real and synthetic datasets in terms of execution time, memory usage and scalability. The percent of reduction in both of execution time and memory are quite high for the sparse datasets but not quite significant for very dense datasets (like Mushroom). On the sparse datasets, the generated trees are much bigger than those on dense datasets. Hence, the bigger is the tree, the more is time cost. Thus, MISFP-growth is more efficient than CFP-growth++ on the whole sparse datasets. With very dense datasets, MISFP-growth and CFP-growth++ work almost same. This is quite expected, as in the case of very dense dataset, most of items are frequent and only few pruning techniques are needed which is the main drawback of CFP-growth++ against MISFP-growth algorithm. For example, in the case of Mushroom dataset, we can observe that the performance efficiency of two compared methods is almost same.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Traditional methods find frequent patterns with a single support threshold. These methods implicitly assume that all items in the database are of the same nature or have similar frequencies in the database. However, this assumption is not valid in the real world applications since some items appear very frequently while others rarely appear in the database. Furthermore, setting an appropriate single *minsup* is a crucial issue since we will encounter the following problem. By mining with high *minsup*, we cannot find interesting patterns involving infrequent items. To produce these patterns, we should set *minsup* very low. In this case, too many meaningless patterns will be generated. To avoid this problem, it is necessary to extract useful patterns involving rare items and reflect the nature of each item in database. The existing methods have been extended to allow the user to specify *minsup* value for each item. By this way, each item in the database can have a minimum item support specified by the user. Therefore, different MIS values for different items enable us to extract different patterns.

Several methods have been proposed to mine frequent patterns with MIS. Recently, a tree-based methods have been proposed to extract frequent patterns under multiple thresholds. However, these methods consume a huge amount of memory and time since they carry out exhaustive search to discard infrequent items and rebuild the tree.

In this thesis, we propose an improved method to extract interesting patterns involving both of frequent and rare patterns. To proficiently discover interesting patterns, this method dynamically assigns appropriate minimum item support threshold to each item instead of using a single minimum support threshold. We propose MISFP-Tree to hold all necessary information that are needed in mining process. This tree is efficiently constructed with only useful items that will potentially play role to generate frequent and rare patterns. Thus, reconstructing the tree is not needed since our proposed tree, MISFP-Tree, is built without any useless items. In this thesis also, MISFP-growth algorithm has been proposed to efficiently mine interesting patterns from MISFP-Tree under multiple support thresholds.

To show the effectiveness of the proposed method, we evaluate its performance by running experiments on both sparse and dense datasets. The experimental results reveal that our proposed method outperforms the recent tree-based, CFP-growth++ method, almost in all cases. The results indicate that MISFP-growth is better than CFP-growth++ in term of both runtime and memory consumption. A set of experiments also confirm that MISFP-growth scales up better than CFP-growth++ as the size of the datasets is increased in both of real and synthetic datasets.

Up to now, a few methods have been proposed to mine frequent patterns with MIS. We can sense that there is much more to do in this field. For upcoming studies, the interesting patterns can be discovered from rare items as frequent patterns indicate the known and the expected information while rare patterns represent unexpected or previously unknown knowledge, which is more interesting to users. Because of this, our method can be extended to find meaningful rare patterns with multiple thresholds without generating a huge number of frequent patterns. In addition, MISFP-growth algorithm can be extended to mine frequent patterns under multiple support thresholds in incremental databases.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami, Mining association rules between sets of items in large databases. *ACM SIGMOD Record* 22, no. 2, 1993, p.207-216.

[2] Rakesh Agrawal, and Ramakrishnan Srikant, Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, p.487-499.

[3] Han, Jiawei, Jian Pei, and Yiwen Yin, Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, vol. 29, no. 2, 2000, p.1-12.

[4] Liu, Yu-Cheng, Chun-Pei Cheng, and Vincent S. Tseng, Discovering relational-based association rules with multiple minimum supports on microarray datasets. *Bioinformatics* 27, no. 22, 2011, p.3142-3148.

[5] Grahne, Gosta, and Jianfei Zhu, Fast algorithms for frequent itemset mining using fp-trees. *Knowledge and Data Engineering, IEEE Transactions on* 17, no. 10, 2005, p.1347-1362.

[6] Jalan Shalini, Anurag Srivastava, and G. K. Sharma, A non-recursive approach for FP-tree based frequent pattern generation. In *Research and Development (SCOReD), IEEE Student Conference on*, 2009, p. 160-163.

[7] Wei Zhang, Hongzhi Liao, and Na Zhao, Research on the FP growth algorithm about association rule mining. In *Business and Information Management, 2008. ISBIM'08. International Seminar on*, vol. 1, 2008, p. 315-318.

[8] Liu Bing, Wynne Hsu, and Yiming Ma, Mining association rules with multiple minimum supports. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999, p.337-341.

[9] Tiantian Xu, and Xiangjun Dong, Mining frequent patterns with multiple minimum supports using basic Apriori. In *Natural Computation (ICNC), 2013 Ninth International Conference on*,2013, p. 957-961.

[10] Kiran, R. Uday, and P. Krishna Re, An improved multiple minimum support based approach to mine rare association rules. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, 2009, p. 340-347.

[11] Tseng, Ming-Cheng, and Wen-Yang Lin, Maintenance of generalized association rules with multiple minimum supports. *Intelligent Data Analysis* 8, no. 4, 2004, p.417-436.

[12] Hu, Ya-Han, and Yen-Liang Chen, Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. Decision Support Systems 42.1, 2006, p.1-24.

[13] Sinthuja, M., S. Sheeba Rachel, and G. Janani, MIS-Tree Algorithm for Mining Association Rules with Multiple Minimum Supports. Bonfring International Journal of Data Mining 1.Special Issue Inaugural Special Issue, 2011, p.1-5.

[14] R. Uday Kiran, and P. Krishna Reddy, Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In *Proceedings of the 14th International Conference on Extending Database Technology*, 2011, p.11-20.

[15] Taktak, Wiem, and Yahya Slimani, MS-FP-Growth: A multi-support Vrsion of FP-Growth Agorithm. *International Journal of Hybrid Information Technology* 7, no. 3, 2014, p.155-166.

[16] Hoque, F. A., M. Debnath, N. Easmin, and K. Rashed, Frequent pattern mining for multiple minimum supports with support tuning and tree maintenance on incremental database. *Research Journal of Information Technology* 3, no. 2, 2011. p.79-90.

[17] Chen, Yi-Chun, Grace Lin, Ya-Hui Chan, and Meng-Jung Shih, Mining Frequent Patterns with Multiple Item Support Thresholds in Tourism Information Databases. In *Technologies and Applications of Artificial Intelligence*, Springer International Publishing, 2014, p.89-98.

[18] Ryang, Heungmo, Unil Yun, and Keun Ho Ryu, Discovering high utility itemsets with multiple minimum supports. *Intelligent data analysis* 18, no. 6, 2014, p.1027-1047.

[19] Lee, Yeong-Chyi, Tzung-Pei Hong, and Wen-Yang Lin, Mining association rules with multiple minimum supports using maximum constraints. *International Journal of Approximate Reasoning* 40, no. 1, 2005, p.44-54.

[20] Han, Jiawei, and Yongjian Fu, Discovery of multiple-level association rules from large databases. In *VLDB*, vol. 95, 1995, p.420-431.

[21] Rice, John A, Mathematical statistics and data analysis, 1995.

[22] Bansal, Anubha, Neelima Baghel, and Shruti Tiwari, An novel approach to mine rare association rules based on multiple minimum support approach. *International Journal of Advanced Electrical and Electronics Engineering,(IJAEEE)* 10, p.75-80.

[23] Szathmary, Laszlo, Amedeo Napoli, and Petko Valtchev, Towards rare itemset mining. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, vol. 1, 2007, p.305-312.

[24] Sadhasivam, Kanimozhi SC, and Tamilarasi Angamuthu, Mining rare itemset with automated support thresholds. *Journal of Computer Science* 7, no. 3, 2011, p.394.

[25] Deng, Zhi-Hong, and Sheng-Long Lv, Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications* 41, no. 10, 2014, p.4505-4512.

[26] Zhang, Wei, Hongzhi Liao, and Na Zhao, Research on the FP growth algorithm about association rule mining. In *Business and Information Management, 2008. ISBIM'08. International Seminar on*, vol. 1, 2008, p. 315-318.

[27] Deng, ZhiHong, ZhongHui Wang, and JiaJian Jiang, A new algorithm for fast mining frequent itemsets using N-lists. *Science China Information Sciences* 55, no. 9, 2012, p.2008-2030.

[28] Liu, Li, Eric Li, Yimin Zhang, and Zhizhong Tang, Optimization of frequent itemset mining on multiple-core processor. In *Proceedings of the 33rd international conference on Very large data bases*, 2007, p.1275-1285.

[29] Olson, David L., and Dursun Delen. Advanced data mining techniques. Springer Science & Business Media, 2008.

[30] Swartz, Nikki. IBM, Mayo Clinic to mine medical data. Information Management Journal 38.6, 2004, p.8-9.

[31] Roche, T. G, Expect increased adoption rates of certain types of EHRs, EMRs. Managed Healthcare Executive 16.4, 2006, p.58.

[32] Havenstein, Heather, IT efforts to help determine election successes, failures: Dems deploy data tools; GOP expands microtargeting use. *Computerworld* 40, no. 45, 2006, p.1.

[33] Weng, Sung-Shun, Ben-Jeng Wang, Ruey-Kei Chiu, and Sheng-Hung Su, The study and verification of mathematical modeling for customer purchasing behavior. *Journal of Computer Information Systems* 47, no. 2, 2006, p.46-57.

[34] Rejesus, Roderick M., Bertis B. Little, and Ashley C. Lovell, Using data mining to detect crop insurance fraud: is there a role for social scientists. *Journal of Financial Crime* 12, no. 1, 2005, p.24-32.

[35] Aggarwal, Charu C., and Jiawei Han, eds, Frequent pattern mining. Springer, 2014.

[36] Aggarwal, Charu C., and Chandan K. Reddy, eds, Data clustering: algorithms and applications. CRC Press, 2013.

[37] Jain, Anil K, Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, no. 8, 2010, p.651-666.

[38] Ma, Bing Liu Wynne Hsu Yiming, Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*. 1998.

[39] Tan, Pang-Ning, M. Steinbach, and V. Kumar, Chapter 6. Association analysis: basic concepts and algorithms. Introduction to Data Mining, 2005.

[40] Zaki, Mohammed J, Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on* 12, no. 3, 2000, p.372-390.

[41] Shenoy, Pradeep, Jayant R. Haritsa, S. Sudarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah, Turbo-charging vertical mining of large databases. In *ACM SIGMOD Record*, vol. 29, no. 2, 2000, p.22-33.

[42] Zaki, Mohammed Javeed, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li, New Algorithms for Fast Discovery of Association Rules. In *KDD*, vol. 97, 1997, p.283-286.

[43] Stanišić, Predrag, and Savo Tomović, Apriori multiple algorithm for mining association rules. *Information Technology And Control* 37, no. 4, 2015.

[44] Yao, Jun, and Yi Lin, Analysis of Aprior algorithm in mining association rules. In *Applied Mechanics and Materials*, vol. 411, 2013, p.1072-1075.

[45] Jin, Kan, An efficient algorithm for association mining, In *Knowledge Acquisition and Modeling, 2009. KAM'09. Second International Symposium on*, vol. 1, 2009, p.291-295.

[46] Agarwal, Ramesh C., Charu C. Aggarwal, and V. V. V. Prasad, Depth first generation of long patterns. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, p.108-118.

[47] Tsang, Sidney, Yun Sing Koh, and Gillian Dobbie, Finding interesting rare association rules using rare pattern tree. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems VIII*, 2013, p.157-173.

[48] Bhatt, Urvi Y., and Pratik A. Patel, An effective approach to mine rare items using Maximum Constraint. In *Intelligent Systems and Control (ISCO), 2015 IEEE 9th International Conference on*, 2015, p.1-6.

[49] Ahmed, Chowdhury Farhan, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, Young-Koo Lee, and Ho-Jin Choi, Single-pass incremental and interactive mining for weighted frequent patterns, *Expert Systems with Applications* 39, no. 9, 2012, p.7976-7994.

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

[51] Burdick, Douglas, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu, Data Mining-MAFIA: A Maximal Frequent Itemset Algorithm. *IEEE Transactions on Knowledge and Data Engineering* 17, no. 11, 2005, p.1490-1504.

[52] Pei, Jian, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongqing Yang. H-mine: Hyper-structure mining of frequent patterns in large databases. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, 2001, p.441-448.

[53] Pavón, Judith, Sidney Viana, and Santiago Gómez. Matrix Apriori: Speeding Up the Search for Frequent Patterns. In *Databases and Applications*, 2006, p.75-82.

[54] Le, Thi, Thi Nguyen, and Tae Chong Chung, BitApriori: an apriori-based frequent itemsets mining using bit streams. In *Information Science and Applications (ICISA), 2010 International Conference on*, 2010, p.1-6.

[55] Ghanem, Ahmed M., and Hamed M. Sallam, Hybrid search based association rule mining. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2011.

[56] Park, Jong Soo, Ming-Syan Chen, and Philip S. Yu, Using a hash-based method with transaction trimming for mining association rules. *Knowledge and Data Engineering, IEEE Transactions on* 9, no. 5, 1997, p.813-825.

[57] Brin, Sergey, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur, Dynamic itemset counting and implication rules for market basket data.In*ACM SIGMOD Record*, vol. 26, no. 2, 1997, p.255-264.

[58] Grahne, Gösta, and Jianfei Zhu, Efficiently using prefix-trees in mining frequent itemsets. In *FIMI*, vol. 90, 2003.

[59] Schmidt-Thieme, Lars, Algorithmic Features of Eclat, FIMI. 2004.

[60] Moriwal, R, FP-growth tree for large and dynamic data set and improve efficiency. *J. Inform. Comput. Sci* 9, 2014, p.83-90.

[61] Sadeq darrab, Belgin Ergenç, Frequent pattern mining under multiple support thresholds. WSEAS transactions on computer research, ISSN / E-ISSN: 1991-8755 / 2415-1521, vol. 4, 2016, p. 1-10.