INCREMENTAL SHARE FREQUENT ITEMSET MINING

CHAYANAN NAWAPORNANAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2018
KMITL-2018-SC-D-002-051

INCREMENTAL SHARE FREQUENT ITEMSET MINING

CHAYANAN NAWAPORNANAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2018
KMITL-2018-SC-D-002-051

หัวข้อวิทยานิพนธ์	การสืบค้นรายการที่เกิดร่วมบ่อยแบบเพิ่มเติมได้
นักศึกษา	นางสาวชญาณ์นันท์ นวพรอนันต์
รหัสประจำตัว	56605008
ปริญญา	ปรัชญาดุษฎีบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2561
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ.ดร.ศรัณย์ อินทโกสุม
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	รศ.ดร.วีระ บุญจริง

บทคัดย่อ

งานวิทยานิพนธ์นี้เสนอขั้นตอนวิธีใหม่ที่มีประสิทธิภาพในการสืบค้นรายการที่เกิดร่วมบ่อยด้วยวิธีการแบบเพิ่มเติมได้ ชื่อว่า Incremental Share Frequent Itemset Mining หรือเรียกแบบสั้นว่า IS-FUP โดยอัลกอริทึม IS-FUP ปรับปรุงมาจากอัลกอริทึม Mining Complete Share-Frequent Itemsets (MCShFI) และใช้เทคนิค Fast Update Concept (FUP) อัลกอริทึมที่เสนอนี้สามารถรองรับการเพิ่มขึ้นของข้อมูลใหม่เพื่อรองรับการขยายตัวอย่างรวดเร็วของข้อมูลในปัจจุบัน โดยสืบค้นรายการที่เกิดร่วมบ่อยจากกลุ่มของทรานแซคชันที่เพิ่มเข้ามาใหม่เป็นหลัก และใช้ประโยชน์จากองค์ความรู้ที่ได้สืบค้นก่อนหน้าจากฐานข้อมูลเดิม จากการทดลองแสดงให้เห็นว่าอัลกอริทึมที่นำเสนอแบบเพิ่มเติมได้นี้สามารถค้นหาองค์ความรู้ได้อย่างครบถ้วน และใช้เวลาในการประมวลผลน้อยกว่าขั้นตอนวิธีแบบเดิมที่ไม่รองรับการเพิ่มขึ้นของข้อมูล นอกจากนี้งานวิทยานิพนธ์ได้นำเสนอตารางพีเอสเทเบิล เป็นตารางที่จัดเก็บกลุ่มของรูปแบบข้อมูลพร้อมด้วยปริมาณข้อมูลของรูปแบบ โดยที่พีเอสเทเบิลสามารถผนวกทรานแซคชันที่มีรูปแบบเดียวกันให้จัดเก็บเพียงหนึ่งทรานแซคชันเท่านั้น พีเอสเทเบิลจะถูกสร้างขึ้นเพียงครั้งเดียวและสามารถรองรับการเพิ่มข้อมูลใหม่ในอนาคตได้โดยไม่จำเป็นต้องสร้างใหม่

คำสำคัญ: การทำเหมืองข้อมูล การสืบค้นรายการที่เกิดร่วมบ่อย วิธีการการแบบเพิ่มเติมได้

Thesis Title	Incremental Share Frequent Itemset Mining
Student	Chayanan Nawapornanan
Student ID	56605008
Degree	Doctor of Philosophy (Computer Science)
Program	Computer Science
Year	2018
Thesis Advisor	Assistant Professor Dr.Sarun Intakosum
Thesis Co-advisor	Associate Professor Dr.Veera Boonjing

Abstract

In this dissertation, a new efficient algorithm for mining share frequent itemset on incremental database is presented, called Incremental Share Frequent Itemset Mining, or IS-FUP, for short. It is extended from the MCShFI algorithm in a batch way with Fast Update Concept (FUP). IS-FUP eliminates the rescanning by mining the share frequent itemset only from new transactions. As a result, the previously discovered share frequent itemset are updated efficiently. Extensive performance analyses show that the execution time of the IS-FUP algorithm outperforms those of the batch way on both synthetic datasets and real-life datasets. Moreover, a new data structure that captures all atomic itemsets with their count information, namely, PSTable Knowledge, is presented. The structure is constructed by scanning the database only once. When new transactions arrive, they can be immediately added to the existing database without reconstructing.

Keywords: Data Mining, Share Frequent Itemset Mining, Incremental Mining

Acknowledgements

During my PhD study, there are some people that I can't thank you enough for their help and worth suggestion throughout my university life.

To my advisors, Associate Professor Dr. Veera Boonjing and Assistant Professor Dr. Sarun Intakosum, I am deeply grateful for the opportunity and worth suggestions from their extensive experience in academic research. In some moments that I failed to move on, they never left me behind and from their advice it lights up to keep me going.

Besides my advisors, I would like to thank the thesis committee: Associate Professor Dr. Jeeraporn Weerapun, Assistant Professor Dr. Krung Sinapiromsaran, Assistant Professor Dr. Nualsawat Hiransakolwong and Assistant Professor Dr. Saichon Jaiyen, for taking their valuable time to insight comments and making this thesis complete.

In addition, I really appreciate to Associate Professor Dr. Chatchai Leenawong, Associate Professor Dr. Ponruedee Netisopakul and Assistant Professor Dr. Supachate Innet for giving me opportunities and valuable advice.

The most important, I would like to acknowledge with gratitude, the support and love of my mother and my brothers. They all kept me going and this dissertation would not have been possible without them.

Besides, I would like to thank you the KMITL for making me meet beloved teachers, great friends, great seniors and great juniors.

Miss Chayanan Nawapornanan

Table of Contents

Page

Table of Contents (Cont.)

Page

List of Tables

Tables **Page**

List of Figures

Figure **Page**

List of Figures (Cont.)

Figure **Page**

Chapter 1
Introduction

1.1 Research Motivation

Association Rule Mining (ARM) is an important topic within the area of data mining. This topic is motivated by an application known as market basket analysis which concerns how to identify relationships among items purchased by customers [1]. In addition, association rules mining is widely employed; for example, in cross marketing [19], new product development [20], personal services, and credit approval assessment for e-business [17]. The process of discovering all the association rules [22] comprises of two steps: 1) Discovering all frequent itemsets that satisfying a minimum support threshold, and 2) Creating all rules from discovered frequent itemsets that meet the confidence threshold. Most of the researchers are interested in solving problems on creating frequent itemset because it is costly and consumes high resources. Several mechanisms can be classified into two types, namely, the level-wise [2,5,26,27,28,31] and the pattern-growth approaches [8,10,11,29,30] discovered to obtain frequent itemsets. In general, the level-wise approach, in each iteration, is divided into two steps, the joining and the pruning. The joining step generates new candidates by combining frequent itemsets from the previous iteration. After that, the pruning step filters out those candidates not satisfying the minimum support threshold. The pattern-growth approach scans the database twice. The first scanning results in a sorted list of all frequent items that pass the minimum support. The second scanning constructs a frequent-pattern tree containing information on occurrence of all itemsets (a word “occurrence” represents presence or absence of an item in a transaction, that is, considering an item in a transaction database as a binary (0/1) value).

However, the frequent itemset mining methods are based on support measure by counting an occurrence of an item in a transaction database as a binary (0/1) value. For example, if a customer purchases 10 units of item A, item A will be counted as 1 rather than its frequency, 10 (a word “frequency” represents a sold quantity of an item in a transaction). In fact, there could be transactions with many units of the same item; therefore, examining only occurrences of itemsets in a

transaction cannot reflect any other implicit factors, such as quantity sold, costs and profits [12]. In dealing with this limitation, several modified and new measures have been proposed. For example, Cai et al. [6] proposed a weighted itemset mining, which modified the support measure by adding another factor called “weight” to obtain better resulted frequent itemsets. The weight can be the importance, the interest, or other aspect of the item to differentiate all items. Apart from the modified support measure, a new measure called share measure was proposed by Carter et al. [4]. This share frequent itemset mining extends traditional frequent itemset mining with consideration of the purchased quantities. Later, Chan et al. [13] proposed a high utility frequent itemset mining, which concerns with not only the purchased quantity but also the unit profit of each item.

In addition, several mining methods were proposed for efficiently discovering share frequent itemsets. The Zero pruning (ZP) and the Zero subset pruning (ZSP) proposed by [7,12] can find all share frequent itemsets using an exhaustive search method. However, both algorithms only prune the generated candidates that have zero number of transaction. A number of mechanisms have been introduced for seeking share frequent itemsets with share infrequent subsets, but they cannot extract the complete set of share frequent itemsets such as Share infrequency pruning algorithm (SIP) [7], Combine all counted algorithm (CAC) [9] and Item add-back algorithm (IAB) [12]. Next, Fast Share Measure (ShFSM) method [14] was introduced to solve the weakness of the existing algorithms by using a level-closure property. However, ShFSM still produces too many candidates in each round and also cannot extract the complete set of share frequent itemsets. After that, Direct Candidates Generation (DCG) [15] was proposed and outperforms ShFSM by generating candidate itemsets without pruning and joining steps in each iteration. DCG can maintain the downward-closure property by using the transaction-measure-value of an itemset (Definition 2.4). The number of candidates is generated when DCG is less than ShFSM but DCG suffers from the level-wise candidate generation-and-test problems and needs several database scans. After that, an efficient algorithm for mining complete share frequent itemsets [25] was proposed, named MCShFI, for efficiently extracting complete share frequent itemsets based on a level-wise generating property. It can reduce the runtimes by using transaction measure

value as an effective upper bound of each item to generate only the promising candidate itemsets in each iteration.

The above methods belong to batch mining, that is, the database is assumed to be a static one and the mining processes are performed in a batch mode. In real-world applications, databases tend to be large and grow over time, so that the previous discovered information become obsolete or some new information may emerge. Therefore, in this research, an incremental share frequent itemset mining will be proposed in order to obtain new frequent itemsets from the updated database using mainly the newly inserted transactions and the original share frequent itemsets resulted from the initial mining on the original database.

1.2 Objectives of the study

This thesis aims at developing algorithm that can effectively handle large growing database. An incremental algorithm called Incremental Share Frequent Itemset Mining, or IS-FUP, for short, that eliminates the rescanning by mining share frequent itemsets only from new transactions thus results in efficiently updating the previous discovered share frequent itemset. In addition, the thesis presents an incremental data structure called "PSTable" to maintain an incremental database.

1.3 Hypothesis of Study

The execution time of the incremental share frequent itemset mining with FUP concept can be reduced better than that of the traditional method in a batch way.

1.4 Scopes of Thesis

The scope of this research is as follows.

1) Develop an incremental share frequent itemset mining algorithm with the FUP concept that can reduce the execution time better than the traditional batch algorithm.

2) The proposed solution was evaluated with standard datasets from Frequent Itemset Mining Dataset Repository (http://fimi.ua.ac.be/data/) and count information was randomly assigned to each item in each transaction, ranging from 1 to 10.

1.5 Organization

This thesis is organized as follows.

Chapter 2 presents the concept of Frequent Itemset mining and Share Frequent Itemset mining, basic definitions and background, and the literature reviews that are related to Share Frequent Itemset mining.

Chapter 3 explains the IS-FUP solution which is an incremental share frequent itemset mining that handles insertion of transactions.

Chapter 4 describes experimental results for both synthetic datasets and real-life dataset to demonstrate of IS-FUP and MCShFI on runtimes and analyzes the size of the PSTable and Frequency table.

Chapter 5 summarizes the proposed solution and discusses limitations and the future work.

Chapter 2
Literature Reviews

This chapter is organized as follows. Section 2.1, the concept of Frequent Itemset mining and the Apriori algorithm are described. In Section 2.2, the concept of Share Frequent Itemset mining, basic definitions and background are given. After that, researches on Share Frequent Itemset Mining and the Fast Update Concept are reviewed in Section 2.3, respectively.

2.1 Frequent Itemset Mining

Frequent Itemset Mining (FIM) is a crucial process and occurs in many real-world applications. These applications include association rule analysis, sequential patterns mining, correlations analysis, multi-dimensional patterns mining. It was introduced in 1993 by Agrawal et al. [1]. Let $I = \{I_1, I_2, ..., I_{m-1}, I_m\}$ be a finite set of items appearing in a transaction database DB. A subset X of I is called an itemset. If an itemset contains k distinct items, then it is defined as the k-itemset. Let $DB = \{T_1, T_2,., T_q,.., T_{n-1}, T_n\}$ be a transaction database where n is the number of transactions in DB and each transaction $T_q \in DB$ must satisfy $T_q \subseteq I$, $(1 \leq q \leq n)$.

In general, a support value of itemset X in the database can be written as Support(X). Support(X) is defined as the ratio of the number of transactions in a database DB containing itemset X. Conceptually, it is the probability that itemset X will be purchased together in DB. If Support(X) is above the *minSup* representing the minimum support threshold specified by a user, then itemset X is called a frequent itemset in DB. For example, suppose that, there are two itemsets: X = {A,B} and Y = {C,D} with their corresponding support values of 70% and 40% and the *minSup* is set at 50%. Therefore, itemset X is determined as the frequent itemset because its Support(X) is greater than the *minSup* while itemset Y is an infrequent itemset because its Support(Y) is lower than the *minSup*.

Apriori algorithm

A classical apriori algorithm is proposed by Agrawal and Srikant [2]. It is one of the most popular algorithms discovering frequent itemsets from a transaction

database by using candidate generation for association rules. Apriori employs an efficient search methodology called, the downward-closure property of support, or the Apriori property, which expresses that the subsets of a frequent itemset are also frequent. Likewise, the supersets of an infrequent itemset are infrequent. The objective of Apriori is to discover all itemsets whose support ratio is larger than the minimum support threshold (*minSup*). The algorithm can be described as follows:

First of all, *k* is set to be 1, each 1-itemset X is loaded into Apriori as a candidate 1-itemset having its Support(X) computing from a transaction database. After that, it determines whether Support(X) is bigger than or equal to the *minSup*. If Support(X) satisfies the condition, keeps it in F_k (frequent itemset in round k) and k is increased by 1. For the next round, in each iteration, it is divided into two steps, the joining and the pruning steps.

The joining step will determine a set of candidate *k*-itemsets denoted C_k from F_{k-1}. The Apriori algorithm assumes that items within a transaction or itemset are sorted in lexicographical order. It does this by joining F_{k-1} with itself. The joining F_{k-1} to F_{k-1} is only performed between itemsets having the first (*k*-2) items in common with each other. Suppose that *k* = 4 and itemset A and B are members of F_3. Itemsets A and B will be joined for generating candidate 4-itemsets when the position 1 and 2 of itemsets A and B are the same. The position 3 of itemset A is less than the position 3 of itemset B, as shown in Figure 2.1.

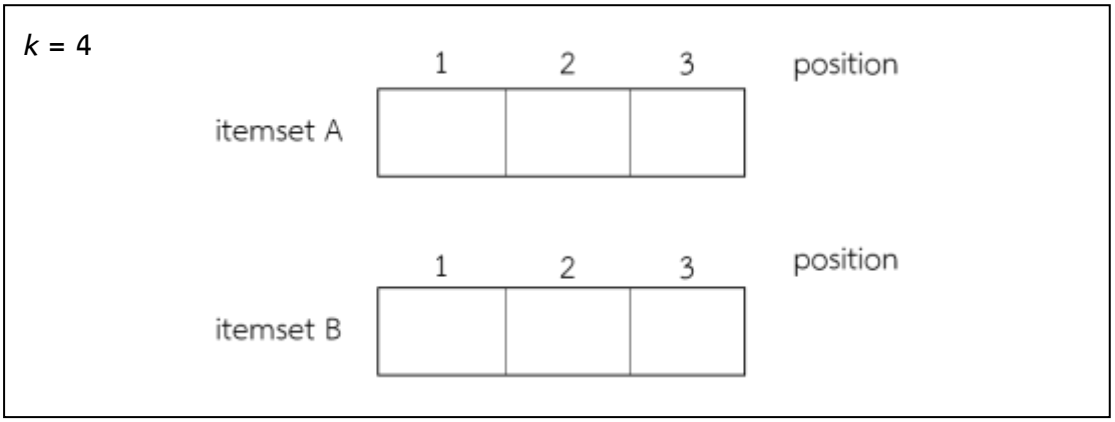


Figure 2.1 An example of the joining step when k = 4

The second step, the pruning step, filters out those candidates whose are not satisfied the *minSup*. At first, the support value of each candidate k-itemset (C_k) X is computed from a transaction database. Then, it determines whether Support(X) is bigger than or equal to the *minSup*. If Support(X) satisfies the condition, keeps it in F_k. The downward-closure property, the Apriori property, is used for reducing the size of C_k. This property states that any (*k*-1)-itemset is infrequent so it cannot be a subset of a frequent *k*-itemset. Therefore, if any (*k*-1)-subset of a candidate k-itemset is not a member in F_{k-1} then it can be removed from C_k because it cannot be frequent. The resulting set of frequent *k*-itemsets is denoted F_k and *k* is increased by 1. The Apriori algorithm will repeat the above step until no new candidate itemset is generated.

2.2 Share Frequent Itemset Mining

From Section 2.1, Frequent Itemset Mining treats all items in a transaction database as binary (0/1) values. It only considers the number of transactions in the database containing desired patterns. The information obtained from Frequent Itemset Mining represents a set of itemsets bought together in a transaction with their support values. For example, suppose that a database containing a set of transactions, half of which items “A” and “B” are bought together, results is a support value of 50%. The Frequent Itemset Mining can be useful for shelf management in promoting two products to be purchased together.

In fact, there could be a transaction with many units of an item. The customers’ purchased database stores the actual quantities of item sold in a transaction. However, these quantities have not been realized in the support mentioned above. In 1997, Carter et al. [4] introduced the Share Frequent Itemset Mining (SFIM) to discover useful knowledge about numerical attributes, such as the quantity sold, associated with an itemset in a transaction. The information obtained from SFIM represents a set of itemsets with high proportions of total quantity sold from a transaction database. For example, suppose that a database contains a set of transactions, some of which items A and B are purchased together. The quantities of items A and B in those transactions sum up to 50% of the quantities of all items purchased in all transactions. This results in a share value of 50% for items “A and B”. SFIM can be useful for stock management in terms of reserving the stored

quantities. In general, a share value of itemset X in the database can be represented by SH(X). A formal definition of SH(X) will later be stated mathematically in Definition 2.8. If SH(X) is above a *minShare* value representing the minimum share threshold specified by a user, then itemset X is called a share frequent itemset in the database. For example, suppose that, there are two itemsets: X = {A,B} and Y = {C,D} with their corresponding share values of 50% and 30% and the *minShare* is set at 45%. Therefore, itemset X is a share frequent itemset because its SH(X) exceeds the *minShare*, while itemset Y is a share infrequent itemset because its SH(Y) is lower than the *minShare*. For extreme cases when SH(X) = 1 where X = {A,B} it means that only items A and B are sold together in every transaction. On the other hand when SH(X) = 0 where X = {A,B} it means that items A and B are not sold together at all in any transaction. Note that, since a share value of itemset is the proportion of the quantity sold of all items in itemset in DB and the total quantity sold in DB, that is, the share value of itemset does not consider the quantity sold of itemset for each item, so that it should be considered all items in itemset together.

Table 2.1 An example of transaction database

Transaction ID	Item sold	Quantity sold
T_1	{A, B, F, G, H}	{A:2, B:8, F:2, G:1, H:6}
T_2	{A, C, E}	{A:5, C:3, E:3}
T_3	{B, C, H}	{B:7, C:2, H:5}
T_4	{C, F, G}	{C:5, F:1, G:1}
T_5	{C, E, F, G}	{C:3, E:1, F:2, G:1}

For illustration purposes, an example of transaction database is provided in Table 2.1 to show that the support value may not be a good measurement in some cases. In each transaction, the list of the item sold along with their quantity sold is given. According to the table, itemset {C,E} occurs in Transactions T_2 and T_5 while {B,H} occurs in Transactions T_1 and T_3, both resulting in the same number of occurrences, 2, and hence the same support value is 2/5. However, the quantity sold of all items in itemset {C,E} in Transactions T_2 and T_5 equal (3+3), (3+1) or 6, 4, respectively. Thus, the total quantity sold of itemset {C,E} in this database = 6+4 = 10. The quantity sold of all items in itemset {B,H} in Transactions T_1 and T_3 equal

(8+6), (7+5) or 14, 12, respectively. Thus, the total quantity sold of itemset {B,H} in this database = 14+12 = 26. In this case, itemsets {C,E} and {B,H} have the same support value of 2, leading to an interpretation that items C and E are bought together twice times in the same way as items B and H. In reality, items C and E are sold in the total quantity of 10 while items B and H are sold in the total quantity of 26. Obviously, 10 and 26 are so different that, in some cases, could indicate more useful meanings than their equal occurrence of 2.

Another example to indicate the opposite result. Again according to Table 2.1, itemset {F,G} occurs in Transactions T_1, T_4 and T_5 yielding 3 occurrences or support value of 3/5. The quantity sold of all items in itemset {F,G} equal (2+1), (1+1), (2+1), respectively, which totals to 8. In terms of the number of occurrences or support value of itemset {F,G} is greater share frequent than that of itemset {B,H} due to 3 > 2 (occurrences) or 3/5 > 2/5 (support values). On the other hand, in terms of the number of quantity sold of itemset {B,H} is, in turn, greater share frequent due to 26 > 8 (total quantity sold). In conclusion, besides support value, the idea of total quantity sold used in share value, can be an alternative criterion for selecting frequent itemsets depending on the situation at hand.

Basic Definitions and Background

In this section, formal definitions of related technical terms are provided along with numerical examples. Table 2.2 shows an example of transaction database categorized into two types, namely, the original database (*O*) and the newly inserted transactions (*db+*). Each type contains Transaction ID, item sold in each transaction, and their respective quantity sold. Transaction T_1 - T_5 are in the original database and transaction T_6 - T_9 are in the group of the newly inserted transactions. For example, T_1 contains item sold A, B, F, G, and H and their quantity sold are 2, 1, 2, 1 and 2, respectively. This table will be used in all subsequent numerical examples in this section.

Table 2.2 Another example of transaction database

Type of DB	Transaction ID	Item sold	Quantity sold
The original database	T_1	{A, B, F, G, H}	{A:2, B:1, F:2, G:1, H:2}
(*O*)	T_2	{A, C, E}	{A:5, C:3, E:3}

	T_3	{B, C, H}	{B:3, C:2, H:2}
	T_4	{C}	{C:5}
	T_5	{C, E, F, G}	{C:3, E:1, F:2, G:1}
The newly inserted transactions (*db+*)	T_6	{B, C, H}	{B:4, C:3, H:2}
	T_7	{A, C, D, E, F}	{A:4, C:2, D:1, E:1, F:5}
	T_8	{A, D}	{A:1, D:2}
	T_9	{A, C, E}	{A:2, C:1, E:1}

Definition 2.1 The measure value of item i_p in transaction T_q, denoted by $mv(i_p,T_q)$, is defined as the sold quantity of item i_p in T_q.
For example, from Table 2.2, the measure value of item {A} in transaction T_2 is, *mv*(A, T_2), is equal to its associated quantity sold, namely, 5.

Definition 2.2 The itemset measure value of itemset X in transaction T_q, denoted by $imv(X,T_q)$, is the summation of the measure value of all items in itemset X in transaction T_q. In other words, it is the quantity sold of all items in itemset X in transaction T_q. Mathematically,

$$imv(X,T_q) = \sum_{X \subseteq T_q, i_p \in X} mv(i_p,T_q) \tag{2.1}$$

For example, from Table 2.2, the itemset measure value of itemset {AC} in transaction T_2 is *imv*({AC}, T_2) = *mv*(A, T_2) + *mv*(C, T_2) = 5+3 = 8.

Definition 2.3 The transaction measure value of transaction T_q, denoted by $tmv(T_q)$, is the summation of the measure value of all items appeared in transaction T_q. In other words, it is the total quantity sold of all items in transaction T_q. Mathematically,

$$tmv(T_q) = \sum_{i_p \in T_q} mv(i_p,T_q) \tag{2.2}$$

For example, from Table 2.2, the transaction measure value of transaction T_2 is *tmv*(T_2) = *mv*({A}, T_2) + *mv*({C}, T_2) + *mv*({E}, T_2) = 5 + 3 + 3 = 11.

Definition 2.4 The transaction measure value of itemset *X,* denoted by tmvs(*X*), is the summation of the transaction measure value of all transactions containing

itemset *X*. In other words, it is the total quantity sold of all transactions containing itemset *X*. Mathematically,

$$tmvs(X) = \sum_{X \subseteq T_q \in DB} tmv(T_q) \tag{2.3}$$

For example, from Table 2.2, *tmvs*({AC}) = *tmv*(T_2) + *tmv*(T_7) + *tmv*(T_9) = 11 + 13 + 4 = 28.

Definition 2.5 The local measure value of itemset *X*, denoted by *lmv*(*X*), is the summation of the itemset measure value of all items in itemset *X* in all transactions containing itemset *X*. In other words, it is the total quantity sold of all items in itemset X in all transactions. Mathematically,

$$lmv(X) = \sum_{T_q \in DB_x} imv(X, T_q) \tag{2.4}$$

where DB_X = { $T_q \in DB \mid X \subseteq T_q$ }.

For example, from Table 2.2, *lmv*({AC}) = *imv*({AC},T_2) + *imv*({AC},T_7) + *imv*({AC},T_9) = 8 + 6 + 3 = 17.

Definition 2.6 The total measure value of a database, denoted by *TMV*(*DB*), is the summation of the transaction measure value of all transactions in the database *DB*. In other words, it is the total quantity sold in the entire database. Mathematically,

$$TMV(DB) = \sum_{T_q \in DB} tmv(T_q) \tag{2.5}$$

and by Definition 2.3, thus,

$$TMV(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q) \tag{2.6}$$

For example, from Table 2.2, The total measure value of the database is *TMV*(*O*∪*db+*) = 8+11+7+5+79+13+3+4 = 67.

Definition 2.7 The share value of itemset *X* in a database *DB*, denoted by *SH*(*X*), is the ratio of the local measure value of *X* to the total measure value of *DB*. In other words, it is the proportion of the quantity sold of all items in itemset *X* and the total quantity sold in the entire database. Mathematically,

$$SH(X) = \frac{lmv(X)}{TMV(DB)} \tag{2.7}$$

For example, from Table 2.2, *SH*({AC}) = lmv({AC})/TMV(DB) = 17/67 = 0.2537. Notice that $0 \le SH(X) \le 1$ because $0 \le lmv(X) \le TMV(DB)$.

Definition 2.8 The minimum share threshold, denoted by *minShare*, is a proportion or a percentage specified by a user to be set as a threshold.

Definition 2.9 Given a *minShare,* if the share value of itemset *X*, *SH*(*X*), is greater than *minShare,* then itemset *X* is a share frequent itemset.
For example, from Table 2.2, if *minShare* set at 0.25, then itemset {AC} is a share frequent itemset because *SH*({AC}) = 0.2537.

Definition 2.10 Given a *minShare*, the minimum local measure value, denoted by *minLmv*, is the minimum quantity sold to be set as a threshold. Mathematically, it is then defined as a ceiling of a product of *minShare* and the total measure value of the database, that is,

$$\lceil \min Share \times TMV(DB) \rceil \tag{2.8}$$

For example, from Table 2.2, given *minShare* = 0.25 and TMV(DB) = 67 from the example in Definition 2.6, then *minLmv* = $\lceil 0.25 \times 67 \rceil$ = $\lceil 16.75 \rceil$ = 17. For any itemset *X*, if *lmv*(*X*) ≥ *minLmv*, then itemset *X* is a share frequent itemset.

Proposition 2.1. The transaction measure value of an itemset *X* maintains the property of downward-closure.
Proof. Let itemset {A} be a share frequent itemset in $DB_{\{A\}}$, where $DB_{\{A\}}$ is a set of all transactions in DB containing {A}. Let itemset {B} be a super itemset of {A}, therefore {B} cannot present in any transactions that {A} is absent. Thus, referring to Definition 2.4, the maximum transaction measure value of {B} is *tmvs*({A}). Therefore, if *tmvs*({A}) is less than *minLmv*, {B} cannot be a share frequent itemset and obviously cannot have a high transaction measure value itemset (including {A}).

By Proposition 2.1, if itemset X is not high transaction measure value itemset, its supersets are not high transaction measure value itemsets either. From Table 2.2 database with *minShare* = 0.25, we have *minLmv* = 17 and *tmvs*({F}) = 13 < *minLmv*. Therefore, all supersets of {F} are not high transaction measure value itemsets.

Definition 2.11 An itemset *X* of a database DB is a high transaction measure value itemset with respect to a threshold, if *tmvs*(*X*) ≥ *minLmv*.

2.3 Literature Reviews

2.3.1 Share Frequent Itemset Mining

In 1997, Carter et al. [4] introduced the share-confidence model to discover useful knowledge about numerical attributes associated with itemsets in a transaction. This numerical attribute reflects all the profit, the cost of itemsets, and associates to the transaction itemsets. Several mining techniques have been proposed for efficiently discovering share frequent itemset that can be divided into two groups. The first group consists of techniques that can extract the complete set of share frequent itemsets from the transaction database, but it takes an exponential running time. The second group consists of techniques that can determine solutions for mining all share frequent itemsets; however, these methods cannot extract the complete set of share frequent itemsets since their models do not satisfy the downward-closure property. Each group will be described below.

The first group consists of Zero pruning (ZP) and Zero subset pruning (ZSP) are proposed by [7] and [12] respectively. Barber and Hamilton [7] proposed an algorithm, named ZP, for discovering the share frequent itemset in the exhaustive search way. It determines all possible itemsets from a transaction database. Any itemsets that have zero number of transactions will be removed. In other words, it keeps only the itemsets that have the total number of transactions greater than zero as candidates. Therefore, the algorithm can extract the complete set of share frequent itemsets from the transaction database but it takes an exponential running time. Although ZP can extract the complete set of share frequent itemsets, it still generated a huge number of unpromising candidates in mining process. Later, an efficient algorithm for exploring all share frequent itemsets [12] was introduced, namely ZSP algorithm, to reduce the number of generated candidates in the mining process by pruning all supersets of any itemsets X that its total number of transactions (TC_X) equals to the value of 0. Assuming that the *2*-itemset {AB} is pre-generation pruned since $TC_{\{AB\}}= 0$, therefore, all supersets of {AB} are not generated as candidate itemsets. ZSP takes more running time and generates a large number of useless candidates in mining process for extracting the complete set of share frequent itemsets from the transaction database.

The second group includes SIP [7], CAC [9], IAB [12] and ShFSM [14]. An algorithm called SIP [7] is introduced to extract share frequent itemset in level-wise

way. This method adopts the downward-closure property by using the share value of itemset (Definition 2.7) to decrease the number of generated candidates in mining process. It starts with reading all *1*-itemsets with their share value. Then, it has to make decision whether the share value of any *1*-itemsets is larger or smaller than the user specified minimum share threshold. If it is larger, it is kept as *1*-candidate for generating candidates of length *2* and so on. If it is smaller, it is not selected and cut off in sequence. This method, however, cannot extract the complete set of share frequent itemsets since this measure does not maintain the property of downward-closure. Later, an efficient algorithm [9], called CAC, was proposed and defined PSH (Predict Share) model for reducing the number of unpromising candidates. CAC generates all *2*-itemsets from all itemsets of length *1* and then calculates their PSH values. For any *2*-itemsets having more PSH value than a user-specified minimum share threshold, they are inserted into *2*-candidates for generate-and-test in the next round. However, CAC cannot extract the complete set of share frequent itemsets since its model does not satisfy the downward-closure property. Next, Barber and Hamilton [12] further proposed the Item add-back algorithm (IAB), that is improved from CAC. It generates all *2*-itemsets from any *1*-itemsets that is having more share values than a defined minimum share threshold. After that, the method computes a PSH value of all *2*-itemsets and keeps only account that has no less PSH value than the certain threshold to the *2*-candidates. For the next round, *k*-candidates are generated by joining the previous candidate (C_{k-1}) with a single itemset in C_1. Nevertheless, the models cannot rely on the property of downward-closure, therefore, the set of share frequent itemsets cannot be discovered completely. Later, Fast Share Measure (ShFSM) was proposed by Li et al. [14] to improve the previous approaches by using a property of level closure to prune the unpromising candidates in the mining process. For *1*-candidates, the method generates all *1*-itemsets with their CF values and then removes unwanted itemsets with lower CF values than the defined threshold. For *k*-candidates, all *k*-candidates are produced from all candidates of length *k*-1 and have more CF values than a given minimum threshold. However, this model does not hold the property of downward-closure, therefore, the set of share frequent itemsets cannot be discovered completely. Furthermore, it still generates too many candidate itemsets and takes more running time.

After that, Li et al. [15] introduces an efficient share frequent itemset mining called "DCG" that relies on the downward-closure property by using the transaction measure value of an itemset (Definition 2.4) to decrease the number of useless candidates. This method adopts the level-wise candidate generation-and-test methodology that directly generates candidates without joining and pruning steps in each pass. DCG outperforms the existing methods [7,9,12,14] in terms of the number of generated candidates and execution time. Moreover, this method extracts the complete set of share frequent itemsets. However, DCG requires more execution time to generate-and-test a large number of unwanted candidates in the mining process since it reads all *1*-itemsets from the transaction database and retains them as *1*-candidate for generating all *2*-candidate. That is, all *1*-candidate contain high-transaction-measure-value and low-transaction-measure-value. For the next round, the method produces the *k*-candidates (C_k) by joining the previous candidate (C_{k-1}) that has their transaction measure value beyond a certain threshold with a single itemset in C_1.

As stated above, reducing the number of candidates and the computation time are an important issue of mining share frequent itemsets. Nawapornanan and Boonjing [25]; therefore, introduced an efficient method for Mining Complete Share frequent Itemsets (MCShFI) to solve the weakness of DCG. MCShFI uses transaction measure value (Definition 2.4) as an effective upper bound of each itemset to reduce the computation time by moving out the number of unwanted candidates at the first round. Furthermore, the generated candidate itemsets of MCShFI are created from the combination of only itemsets with their transaction measure value beyond a certain threshold. As a result, the MCShFI approach is more efficient than the DCG method in terms of execution time. MCShFI is illustrated in Figure 2.2 and can be explained as follows:

First of all, Frequency table is constructed in order to store all transactions and their quantities without considering duplicate patterns. After that, MCShFI starts with determination of TMV(DB) and *minLmv* from Frequency table by Definition 2.6 and 2.10, respectively. Each 1-itemset X is loaded into the MCShFI algorithm and then MCShFI calculates its tmvs(X), lmv(X) and SH(X) from Frequency table by Definition 2.4, 2.5 and 2.7, respectively. After that, it determines whether SH(X) is larger than or equal to the *minShare*. If SH(X) satisfies the condition, stores it to F_k

(share frequent itemset in round k). Then, check whether the tmvs(X) is larger than or equal to the *minLmv*. If tmvs(X) satisfies the condition, keep it to C_k. Increment k by one and then the k-itemsets will be created with C_{k-1}. Repeat the above step until no new candidate itemset is generated.

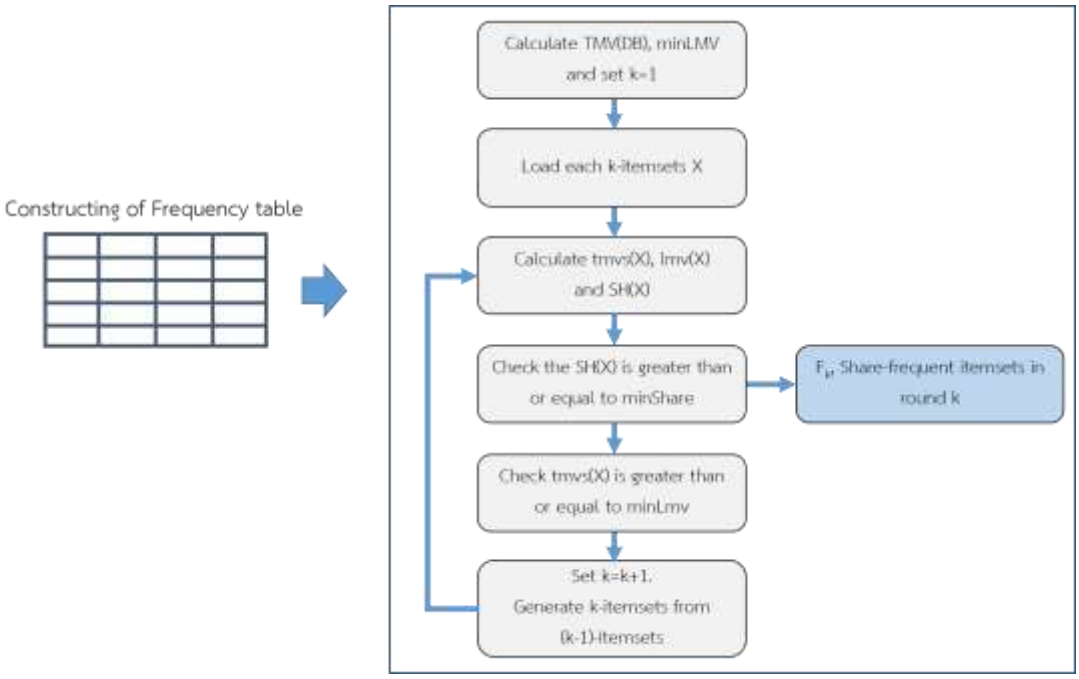


Figure 2.2 an Overview of the MCShFI algorithm

2.3.2 The Fast Update Concept

The number of transactions in a database is increasingly growing up; therefore, the evaluation of derived association rules must be reproduced. In fact, some association rules are newly generated while some other rules become obsolete [18]. Every time the new inserted transaction occurs in the database, in the traditional batch mining algorithm, it has to re-process the whole updated database. This process; therefore, consumes lots of execution time and waste existing mined knowledge. To reduce computation time, Cheung et al. [3] had introduced the Fast Update Algorithm (FUP). The FUP considers only mining results of an original database and new transactions. There are four cases under consideration as shown in Figure 2.3.

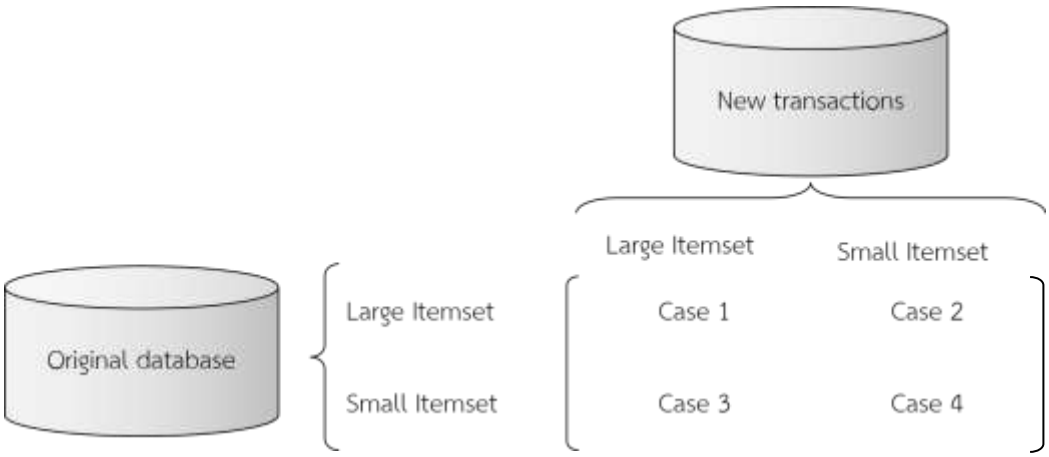


Figure 2.3 Four update cases

Case 1: Both itemset in the original database and the new transaction are huge. Case 2: Itemset in the original database is large while that of the new transaction is small. Case 3: Itemset in the original database is small while that of the new transaction is huge. Case 4: Both itemset in the original database and in the new transaction are small. For Case 1, both the original database and the new transactions are large (frequent). Thus, the weighted average of the counts will be large. Identically, for Case 4, both small (infrequent) original database and small (infrequent) newly inserted transaction yields small weighted average of the counts. This implies that, both Case 1 and Case 4 will not influence the final frequent itemsets. Nevertheless, for Case 2 and Case 3, the weighted average of the counts can be small (infrequent), thus, affects the final frequent itemsets. Hence, to achieve the correct solution, the existing large (frequent) itemsets may be removed for Case 2 while it must be inserted for Case 3.

Chapter 3

Incremental Share Frequent Itemset Mining

This chapter is organized as follows. In Section 3.1, an overview of the IS-FUP algorithm is presented. Then, Section 3.2 explains an efficient data structure to maintain an incremental database, called “PSTable”. A notation is described in Section 3.3. In Section 3.4, the concept of an incremental algorithm that handles insertion of transactions, named IS-FUP, is presented. An illustrative example of IS-FUP is given in Section 3.5. Last section, theorems and proofs of Case 1 and Case 4 are given to prove that the IS-FUP algorithm can mine share frequent itemsets completely and correctly.

3.1 An overview of the IS-FUP algorithm

In this section, an overview of the IS-FUP algorithm is presented. An input of this algorithm consists of PSTable (an original database), newly inserted transactions, a set of itemsets with high transaction-measure-value in the original database and a minimum share threshold defined by users. After IS-FUP is complete, then, a set of itemsets with high transaction-measure-value in the entire updated database will be returned. The overview of the IS-FUP algorithm is shown in Figure 3.1.

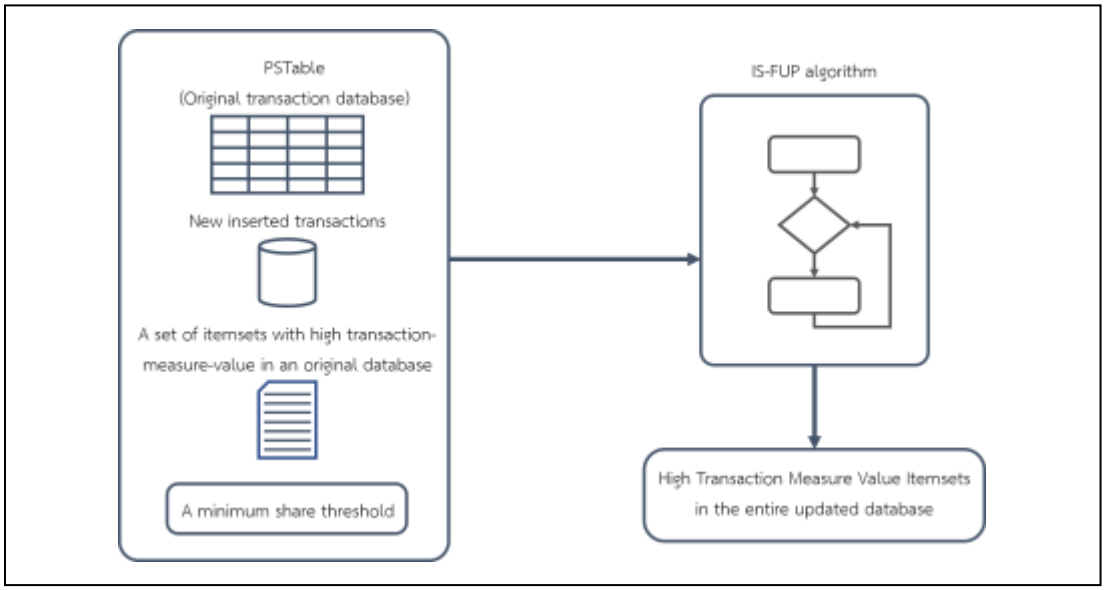


Figure 3.1 An overview of the IS-FUP algorithm

3.2 Maintenance of PSTable

A PSTable is a set of distinct patterns with their count information and it is improved by the concept of a bittable structure [16,21,23,24,25]. This structure can aggregate the transactions that have similar patterns. In addition, the PSTable will be constructed once and it accommodates modified data in the future (do not need to rebuild). The maintaining algorithm of the PSTable is described as below.

Input :

1. The PSTable
2. The original transaction database (O) or the newly inserted transactions (db+)

Output : The updated PSTable

Step 1 : Each transaction from the original database (O) or the newly inserted transactions (db+) is loaded and treated as a pattern X.

Step 2 : The pattern X of the transaction is checked whether it appears in the PSTable or not.

Step 3 : The pattern X appears in the PSTable.

Step 3-1 : The quantity of the transaction is summed with the new entries to gain new value of Total Count column, and the Pattern Count column is incremented by 1.

Step 4 : The pattern X does not appear in the PSTable.

Step 4-1 : The pattern X will be inserted into the PSTable with the quantity of the transaction (Total Count column) and the algorithm sets the Pattern Count column of the pattern X to be 1.

Step 5 : The algorithm repeats the above step for all transactions in the original database or the newly inserted transactions.

Example 1. To illustrate the maintaining process of the PSTable, a database shown in Table 2.2 is used as an example.

The first transaction T_1 = {A:2, B:1, F:2, G:1, H:2} is loaded into the algorithm, that is, {A,B,F,G,H} is treated as a new pattern. Next, the new pattern is checked in the PSTable. The new pattern is new, it will be added to the PSTable and the algorithm sets the Total Count column of the new pattern as the sum of the

quantity of transaction (=8) and also sets the value 1 to the Pattern Count column of the new pattern. The algorithm does the same step to T_5 = {C:3, E:1, F:2, G:1} and the result is illustrated in Table 3.1. After that the next transaction T_6 = {B:4, C:3, H:2} is read to the algorithm, the new pattern of T_6 is {B,C,H} and it is examined in the PSTable. The new pattern exists, the Total Count column is concluded with the new entries to gain new value of total count as (7+9 = 16) and the Pattern Count column is increased by one (1+1 = 2) as shown in Table 3.2. This process is repeated for all transactions and the final PSTable is presented in Table 3.3.

Table 3.1 The result of maintaining algorithm after transaction T_1-T_5 is added

PID	A	B	C	D	E	F	G	H	Total Count	Pattern Count
P1	X	X				X	X	X	8	1
P2	X		X		X				11	1
P3		X	X					X	7	1
P4			X						5	1
P5			X		X	X	X		7	1

Table 3.2 The result of maintaining algorithm after transaction T_6 is added

PID	A	B	C	D	E	F	G	H	Total Count	Pattern Count
P1	X	X				X	X	X	8	1
P2	X		X		X				11	1
P3		**X**	**X**					**X**	**16**	**2**
P4			X						5	1
P5			X		X	X	X		7	1

Table 3.3 The result of the maintaining algorithm

PID	A	B	C	D	E	F	G	H	Total Count	Pattern Count
P1	X	X				X	X	X	8	1
P2	X		X		X				15	2
P3		X	X					X	16	2

P4			X						5	1
P5			X		X	X	X		7	1
P6	X		X	X	X	X			13	1
P7	X			X					3	1

3.3 Notation

I	a set of m items, $I = \{I_1, I_2, ..,I_j,.. , I_m\}$ be a set of literals with counting attributes, where $I_1, I_2, ..,I_j,.. , I_m$ are all items.
X	an itemset, where $X \subseteq I$.
DB	a transaction database, $DB = \{T_1, T_2,.. ,T_n\}$ be a set of transactions, where n is the number of transactions in DB and each transaction $T_q \in DB$ must satisfy $T_q \subseteq I$, $(1 \leq q \leq n)$. The type of DB can be divided into two groups: O and $db+$.
O	the original transaction database.
$db+$	the set of the newly inserted transactions.
U	the entire updated database, $O \cup db+$
C_k	a candidate share frequent itemsets of size k
$minLmv^{db+}$	The minimum local measure value in new transactions $db+$
$minLmv^{U}$	The minimum local measure value in the entire updated database U
TMV^{O}	The total measure value of the original transaction database O
TMV^{db+}	The total measure value of the newly inserted transactions $db+$
TMV^{U}	The total measure value of the entire updated database U
$Htmvs^{O}$	The set of high transaction measure value itemsets in the original transaction database O
$Htmvs^{U}$	The set of high transaction measure value itemsets in the entire updated database U
$Htmvs_k^O$	The set of high transaction measure value k-itemsets in the original transaction database O
$Htmvs_k^{db+}$	The set of high transaction measure value k-itemsets in the newly inserted transactions $db+$
$Htmvs_k^U$	The set of high transaction measure value k-itemsets in the entire updated transactions U

$tmvs^{O}(X)$ The transaction measure value of an itemset X of all transactions in the original transaction database O

$tmvs^{db+}(X)$ The transaction measure value of an itemset X of all transactions in the newly inserted transactions $db+$

$tmvs^{U}(X)$ The transaction measure value of an itemset X of all transactions in the entire updated transactions U

$lmv^{O}(X)$ The local measure value of all transactions in the original transaction database O that containing X

$lmv^{db+}(X)$ The local measure value of all transactions in the newly inserted transactions $db+$ that containing X

$lmv^{U}(X)$ The local measure value of all transactions in the entire updated transactions U that containing X

3.4 The concept of IS-FUP

Assuming that the PSTable, as mentioned in Section 3.2, is built in advance from the original database before the new transactions come and the large itemsets have been already derived from the initial database. In IS-FUP, itemsets are separated into four scenarios according to whether they are large or small itemsets for both in the original database and in the new transactions. To determine whether the itemset is large or small, the value of *tmvs* is calculated (by Definition 2.4). As mentioned earlier, this value contains the properties of downward-closure. If the value of *tmvs* of any itemset is larger than or equal to the minimum local measure value (*minLmv*), it is in the set of high transaction measure value itemsets and in the group of large itemsets. Otherwise, it is in the group of small itemsets and in the set of low transaction measure value itemsets.

Considering the original database and the new transactions that will be inserted, the following four cases of itemsets and their results are given in Table 3.4. Unsurprisingly, an itemset in the updated database will be large when both the itemset in the original database and in the new transaction are large (Case 1). In the same way, it will be small when that of the original database and the new transactions are small (Case 4). This means that, both Case 1 and Case 4 will not affect the final share frequent itemsets, so, the proofs of two cases will be given in Theorems 3.1 and 3.2, respectively.

Table 3.4 Four possible cases of itemsets

Cases : Original-New transactions	**Results**
Case 1: Large itemset - Large itemset	Always large itemset; not considered in the IS-FUP algorithm
Case 2: Large itemset - Small itemset	Determined by rescanning the newly inserted transactions
Case 3: Small itemset - Large itemset	Determined by rescanning the PSTable
Case 4: Small itemset - Small itemset	Always small itemset; not considered in the IS-FUP algorithm

The detail of the proposed algorithm is described step by step below.

Input:

1) The PSTable (*O*)

2) The newly inserted transactions (*db+*) in which each transaction includes items and their quantities

3) A $Htmvs^{O}$, which is stored a set of high transaction measure value of the original transaction database

4) The total measure value of the original transaction database, TMV^{O}

5) The minimum share threshold, *minShare*

Output:

A $Htmvs^{U}$, that is a set of high transaction measure value of the entire updated database.

Step 1: IS-FUP calculates the total measure value of newly inserted transactions using Definition 2.6 denoted by TMV^{db+}. After that, TMV^{U} is calculated ($TMV^{U} = TMV^{O} + TMV^{db+}$). Next, the algorithm computes $minLmv^{U}$ using Definition 2.10 and sets $k = 1$, where k is used for recording the number of patterns in the itemsets currently being processed.

Step 2: The algorithm generates the candidate k-itemsets then computes their transaction measure value $tmvs^{db+}(X)$ and local measure value $lmv^{db+}(X)$ from the new transactions using Definition 2.4 and 2.5, respectively. After that, it determines

whether the $tmvs^{db+}(X)$ of each *k*-itemset *X* is larger than or equal to $minLmv^{db+}$. If $tmvs^{db+}(X)$ satisfies the above condition, put *X* in the set of high transaction measure value *k*-itemset in the newly inserted transactions, $Htmvs_k^{db+}$.

Step 3: For each *k*-itemset *X*, if it is in the set of high transaction measure value in the new transactions and it also appears in the set of high transaction measure value in the original database, do the following steps (Case 1):

Step 3.1: IS-FUP will keep the itemset *X* into both the $Htmv_k^U$ and C_k. Then, the algorithm also removes it from the $Htmvs_k^O$.

Step 4: For each *k*-itemset *X*, if it is in the set of high transaction measure value in the new transactions ($Htmvs_k^{db+}$) but it does not appear in the set of high transaction measure value in the original database $Htmvs_k^O$, do the following steps (Case 3):

Step 4.1: IS-FUP rescans the original database to determine the $tmvs^O(X)$ and $lmv^O(X)$ of *k*-itemset *X*. After that, the algorithm sets both $tmvs^U(X)$ (= $tmvs^O(X)$ + $tmvs^{db+}(X)$) and $lmv^U(X)$ (= $lmv^O(X)$ + $lmv^{db+}(X)$).

Step 4.2: IS-FUP determines whether the $tmvs^U(X)$ is greater than or equal to $minLmv^U$. If $tmvs^U(X)$ satisfies the above condition, stores it to C_k and also adds it into the $Htmvs_k^U$.

Step 5: For each *k*-itemset *X*, if it appears in the set of high transaction measure value in the original database ($Htmvs_k^O$) but it does not appear in the set of high share frequent itemsets from the new transactions ($Htmvs_k^{db+}$), do the following steps (Case 2):

Step 5.1: IS-FUP reads all *k*-itemsets in the $Htmvs_k^O$. After that, the algorithm computes the $tmvs^{db+}$ and the lmv^{db+} from the PSTable and then it calculates both $tmvs^U(X)$ (= $tmvs^O(X)$ + $tmvs^{db+}(X)$) and $lmv^U(X)$ (= $lmv^O(X)$ + $lmv^{db+}(X)$).

Step 5.2: If the $tmvs^U(X)$ is larger than or equal to $minLmv^U$, stores *X* to both C_k and the $Htmvs_k^U$ and also removes X from the $Htmvs_k^O$.

Step 6: Increase *k* by one and then the algorithm generates the candidate *k*-itemsets from the variable C_{k-1}.

Step 7: IS-FUP repeats the above step until no new candidate itemsets is generated and the $Htmvs^O$ is empty.

3.5 An illustrative example of IS-FUP

This section shows how the IS-FUP algorithm can be used for finding out the share frequent itemset from the entire updated database. From Table 2.2, it is assumed that the transactions T_1- T_5 are the original database and the transaction T_6 - T_9 are newly inserted transactions. Also, the minimum share threshold (*minShare*) is set at 0.25 which is 25% of total quantity. Table 3.5 shows all share frequent itemsets (the set of high transaction measure value) that is mined in advanced from the original database (T_1- T_5).

Table 3.5 The $Htmvs^O$

Itemset *X*	$tmvs^O(X)$	$lmv^O(X)$
{A}	19	7
{B}	15	4
{C}	30	13
{E}	18	4
{F}	15	4
{G}	15	2
{H}	15	4
{A,C}	11	8
{A,E}	11	8
{B,H}	15	8
{C,E}	18	10
{F,G}	15	6
{A,C,E}	11	11

Table 3.6 The initial variables for the algorithm

	TMV	*minLmv*
O	38	10
db+	29	8
U	67	17

Step 1: The IS-FUP algorithm calculates the initial variables at first i.e. TMV^{db+}, $minLmv^{db+}$, TMV^{U} and $minLmv^{U}$, respectively, as shown in Table 3.6. Then, *k* is set at 1.

Step 2: The *k*-itemsets in the newly inserted transactions for {A}, {B}, {C}, {D}, {E}, {F} and {H} are read into the algorithm and then IS-FUP calculates their transaction measure value ($tmvs^{db+}$) and the local measure value (lmv^{db+}) from the new transactions. Takes *k*-itemset {A} as an example to illustrate the process. The *k*-itemset {A} appears in the new transactions T_7, T_8 and T_9 and then the algorithm calculates the $tmvs^{db+}$({A}) as (13+3+4), which is 20 and computes the lmv^{db+}({A}) as (4+1+2), which is 7, respectively. The other *k*-itemsets are calculated in the same way and the result is shown in Table 3.7. After that, it determines whether the $tmvs^{db+}(X)$ of each *k*-itemset is larger than or equal to $minLmv^{db+}$. In this example, all *k*-itemsets satisfy the condition and then put them in the set of high transaction measure value *k*-itemsets in the new transaction ($Htmvs_k^{db+}$).

Table 3.7 All *k*-itemsets are calculated from the new transactions

k*-itemset *X	$tmvs^{db+}(X)$	$lmv^{db+}(X)$
{A}	20	7
{B}	9	4
{C}	26	6
{D}	16	3
{E}	17	2
{F}	13	5
{H}	9	2

Step 3: For each *k*-itemset *X*, it is in both the set of high transaction measure value in the new transactions ($Htmvs_k^{db+}$) and the set of high transaction measure value in the original database ($Htmvs_k^{O}$), do the following steps (Case 1):

Step 3.1: The *k*-itemsets are in the $Htmvs_k^{O}$ and the $Htmvs_k^{db+}$ which are as follows {A}, {B}, {C}, {E}, {F} and {H}. These itemsets are recorded to both the $Htmvs_k^{U}$ and C_k. After that, these itemsets are removed from the $Htmvs_k^{O}$. The result is shown in Table 3.8 and Table 3.9.

Table 3.8 After *k*-itemsets (Case 1) are added in the $Htmvs^{U}$

k*-itemset *X	$tmvs^{U}(X)$	$lmv^{U}(X)$
{A}	39	14
{B}	24	8
{C}	56	19
{E}	35	6
{F}	28	9
{H}	24	6

Table 3.9 After *k*-itemsets (Case 1) are deleted from the $Htmvs^{O}$

k*-itemset *X	$tmvs^{O}(X)$	$lmv^{O}(X)$
{G}	15	2
{A,C}	11	8
{A,E}	11	8
{B,H}	15	8
{C,E}	18	10
{F,G}	15	6
{A,C,E}	11	11

Step 4: For each *k*-itemset *X*, it is in the set of high transaction measure value in the newly inserted transactions, $Htmvs_k^{db+}$, but it does not in the set of high transaction measure value in the original database, $Htmvs_k^{O}$, do the following steps (Case 3):

Step 4.1: The *k*-itemset *X* appears in the $Htmvs_k^{db+}$ but it does not appear in the $Htmv_k^{O}$, which is {D}. The IS-FUP algorithm rescans to calculate the $tmvs^{O}(X)$ and the $lmv^{O}(X)$ from the PSTable. Then, the algorithm computes the $tmvs^{U}(X)$ and the $lmv^{U}(X)$, respectively. Let us take the *k*-itemset {D} as an example to illustrate the process. The $tmvs^{db+}$ of {D} and the lmv^{db+} of {D} in the new transactions are 16 and 3, respectively as shown in Table 3.7. The $tmvs^{O}$ of {D} and the lmv^{O} of {D} in the original database are both 0, which are shown in Table 3.10. The updated transaction measure value and the updated local measure value for the updated database of {D} are calculated as $tmvs^{U}$({D}) (=0+16), which is 16, and lmv^{U}({D}) (=0+3), which is 3, respectively.

Table 3.10 After all *k*-itemsets in case 3 are re-calculated

k-itemset *X*	$tmvs^{db+}(X)$	$lmv^{db+}(X)$	$tmvs^{O}(X)$	$lmv^{O}(X)$	$tmvs^{U}(X)$	$lmv^{U}(X)$
{D}	16	3	0	0	16	3

Step 4.2: IS-FUP determines whether the $tmvs^{U}$ of *k*-itemset *X* is larger than or equal to $minLmv^{U}$. If it satisfies the condition, inserts it into both C_k and the $Htmvs_k^U$. For instance, the *k*-itemset {D} does not satisfy the condition.

Step 5: For each *k*-itemset *X*, it does not in the set of high transaction measure value in the new transactions ($Htmvs_k^{db+}$), but it appears in the set of high transaction measure value in the original database ($Htmvs_k^O$), do the following steps (Case 2):

Step 5.1: The IS-FUP algorithm reads all *k*-itemsets from the $Htmvs_k^O$ in Table 3.9, which is {G}. The $tmvs^{db+}$ and the lmv^{db+} of *k*-itemset {G} in the new transactions are calculated, which are both 0. Then, the updated transaction measure value and the updated local measure value for the updated database of {G} are computed as $tmvs^{U}$ ({G}) (=0+15), which is 15, and lmv^{U} ({G}) (=0+2), which is 2, respectively. The result is shown in Table 3.11.

Table 3.11 After all *k*-itemsets in case 2 are re-calculated

k-itemset *X*	$tmvs^{O}(X)$	$lmv^{O}(X)$	$tmvs^{db+}(X)$	$lmv^{db+}(X)$	$tmvs^{U}(X)$	$lmv^{U}(X)$
{G}	15	2	0	0	15	2

Step 5.2: IS-FUP determines whether the $tmvs^{U}$ of {G} is larger than or equal to $minLmv^{U}$. If it satisfies the condition, stores it into both C_k and the $Htmvs_k^U$ and also removes it from the $Htmvs_k^O$. For instance, the *k*-itemset {G} does not satisfy the condition and then {G} is removed from the $Htmvs_k^O$. The result is shown in Table 3.12.

Table 3.12 After *k*-itemsets (Case 2) are deleted from the $Htmvs^{O}$

k-itemset *X*	$tmvs^{O}(X)$	$lmv^{O}(X)$
{A,C}	11	8
{A,E}	11	8
{B,H}	15	8
{C,E}	18	10

{F,G}	15	6
{A,C,E}	11	11

Step 6: Set k = 2 (k + 1). The candidate k-itemsets are generated from C_{k-1}, which are {A,B}, {A,C}, {A,E}, {A,F}, {A,H}, {B,C}, {B,E}, {B,F}, {B,H}, {C,E}, {C,F}, {C,H}, {E,F}, {E,H}, {F,G} and {F,H}.

Step 7: Step 2 - 6 are then repeated until no candidate itemset is generated and the $Htmvs^O$ is empty as shown in Table 3.14. The IS-FUP algorithm returns a set of high transaction measure value in the entire updated database, $Htmvs^U$, that contains 3 share frequent itemsets i.e., {C}, {A,C} and {A,C,E} that are shown in Table 3.13.

Table 3.13 The $Htmvs^U$ after the IS-FUP completed

itemset *X*	$tmvs^U(X)$	$lmv^U(X)$	*SH(X)*
{A}	39	14	0.2090
{B}	24	8	0.1194
{C}	**56**	**19**	**0.2836**
{E}	35	6	0.0889
{F}	28	9	0.1343
{H}	24	6	0.0896
{A,C}	**28**	**17**	**0.2537**
{A,E}	28	16	0.2388
{A,F}	21	13	0.1940
{B,H}	24	14	0.2090
{C,E}	35	15	0.2239
{C,F}	20	12	0.1791
{E,F}	20	9	0.1343
{A,C,E}	**28**	**22**	**0.3284**
{C,E,F}	20	14	0.2090

Table 3.14 The $Htmvs^O$ after the proposed algorithm completed

itemset *X*	$tmvs^O(X)$	$lmv^O(X)$
- Empty -		

3.6 Proofs of Case 1 and Case 4

The following theorems and proofs are given to prove Case 1 and Case 4 in order to show that the IS-FUP solution can discover share frequent itemset completely.

Theorem 3.1. If an itemset *X* is large in both the original database and the newly inserted transactions, then the itemset *X* will be large in the entire updated database (Case 1).

Proof. Assuming, by contradiction, that itemset *X* is not large in the entire updated database, i.e.,

$$tmvs(X)^{O} + tmvs(X)^{db+} < minShare \times (TMV^{O} + TMV^{db+}) \quad (3.1)$$

and so

$$tmvs(X)^{O} + tmvs(X)^{db+} - minShare \times TMV^{db+} < minShare \times TMV^{O} \quad (3.2)$$

By hypothesis, an itemset *X* is large in both the original database and the newly inserted transactions.

it is obtained that

$$tmvs(X)^{O} \geq minLmv^{O} (= minShare \times TMV^{O}) \quad (3.3)$$

and

$$tmvs(X)^{db+} \geq minLmv^{db+} (= minShare \times TMV^{db+}) \quad (3.4)$$

Since

$$tmvs(X)^{O} < tmvs(X)^{O} + tmvs(X)^{db+} - minShare \times TMV^{db+} \quad (3.5)$$

From (3.2) and (3.5), we can conclude that

$$tmvs(X)^{O} < minShare \times TMV^{O} \quad (3.6)$$

(3.6) contradicts (3.3), hence the proof is complete. □

Theorem 3.2. If an itemset *X* is small in both the original database and the newly inserted transactions, then the itemset *X* will be small in the entire updated database (Case 4).

Proof. Assuming, by contradiction, that itemset *X* is not small in the entire updated database, i.e.,

$$tmvs(X)^{O} + tmvs(X)^{db+} \geq minShare \times (TMV^{O} + TMV^{db+}) \quad (3.7)$$

and so

$$tmvs(X)^{O} + tmvs(X)^{db+} - minShare \times TMV^{db+} \geq minShare \times TMV^{O} \quad (3.8)$$

By hypothesis, an itemset X is small in both the original database and the newly inserted transactions.

it is obtained that

$$tmvs(X)^{O} < minLmv^{O} (= minShare \times TMV^{O}) \quad (3.9)$$

and

$$tmvs(X)^{db+} < minLmv^{db+} (= minShare \times TMV^{db+}) \quad (3.10)$$

Since

$$tmvs(X)^{O} > tmvs(X)^{O} + tmvs(X)^{db+} - minShare \times TMV^{db+} \quad (3.11)$$

From (3.8) and (3.11), we can conclude that

$$tmvs(X)^{O} \geq minShare \times TMV^{O} \quad (3.12)$$

(3.9) contradicts (3.12), hence the proof is complete. □

Chapter 4
Results and Discussions

This chapter is organized as follows. In Section 4.1, the experimental environment and datasets are described. In Section 4.2, the IS-FUP algorithm and the MCShFI algorithm are demonstrated by applying to three datasets. Analysis of the PSTable and Frequency table are given in Section 4.3.

4.1 Experimental Environment and Datasets

All algorithms were implemented in Microsoft C++ 2012 and running on a 2.60 GHz Intel(R) Core i7-4720HQ with 12 GB main memory on the Windows 8.1 operating system. Similar to the performance evaluation of the previous share-frequent itemset mining algorithm [15,25,32], the count information was assigned to each item in each transaction, ranging from 1 to 10. Three datasets are used in the experiments and they are acquired from FIMI Repository Page [33]. The characteristics of the datasets are shown in Table 4.1, *m* is the total number of transactions in the dataset and *n* is the number of distinct items in the dataset.

Table 4.1 Characteristics of the experiment datasets

Dataset	*m*	*n*	Type
T10I4D100K	100,000	870	Synthetic
T40I10D100K	100,000	942	Synthetic
Pumsb*	49,046	2,088	Real-life

4.2 The IS-FUP Algorithm Efficiency

In this section, the efficiency of the algorithm is demonstrated by applying IS-FUP to two synthetic datasets (T10I4D100K and T40I10D100K) and a real-life dataset (pumsb*). For more information on synthetic datasets, T10I4D100K and T40I10D100K, the parameters of these synthetic datasets are given as follows: T = the average size of the transactions, and, I = the average size of the maximal frequent itemsets, and, D = the number of transactions. The algorithm efficiency is determined by the processing time and compared to MCShFI [25] in the batch way.

4.2.1 Experimental results on synthetic datasets

The effectiveness of the IS-FUP algorithm in incremental mining and batch MCShFI was tested with T10I4D100K. The minimum share threshold was set at 0.01% and 0.03%, respectively. Both the PSTable and the Frequency table are firstly constructed with 70,000 transactions from T10I4D100K, then, share frequent itemset of the original database is mined from that constructed PSTable. The numbers of inserted transactions were varied from 2,000 to 10,000 and the results are shown in Figure 4.1 and Figure 4.2. For T40I10D100K, experiments were conducted similarly to the previous experiment, but the minimum share threshold was set at 0.1% and 0.3%, respectively. The experimental results are illustrated in Figure 4.3 and Figure 4.4.

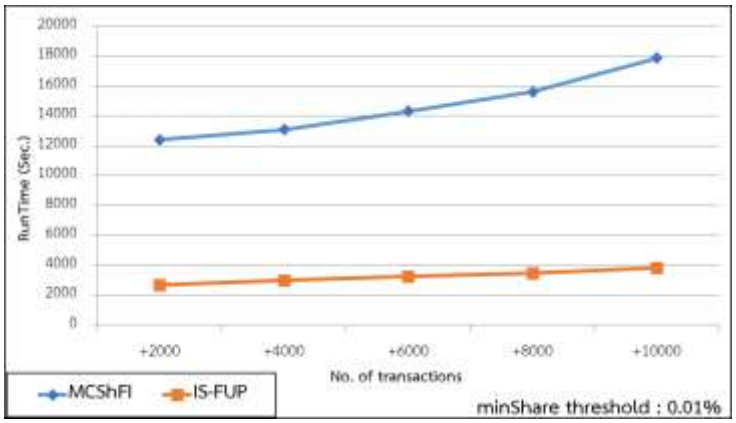


Figure 4.1 The comparison of execution time for various numbers of inserted transactions on T10I4D100K with minShare 0.01%

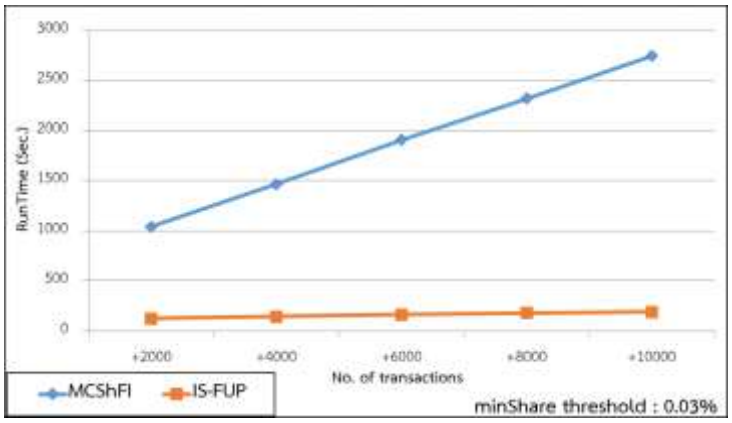


Figure 4.2 The comparison of execution time for various numbers of inserted transactions on T10I4D100K with minShare 0.03%

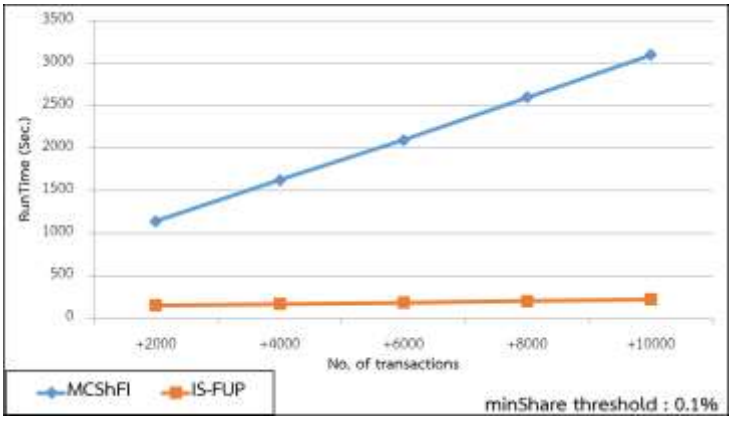


Figure 4.3 The comparison of execution time for various numbers of inserted transactions on T40I10D100K with minShare 0.1%

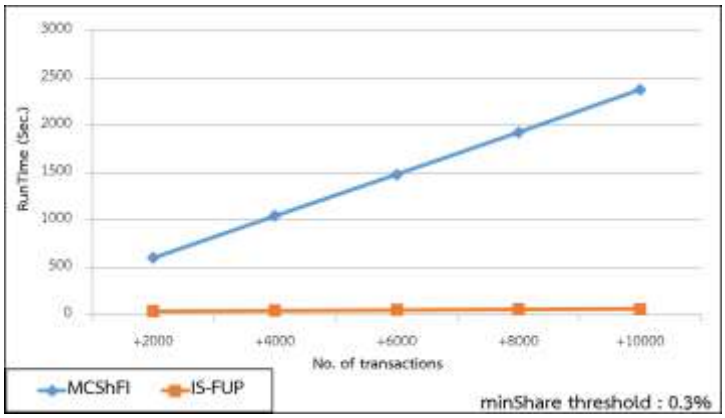


Figure 4.4 The comparison of execution time for various numbers of inserted transactions on T40I10D100K with minShare 0.3%

It can be easily observed from these figures that both algorithms required more execution time when the certain threshold is low. This is reasonable because the algorithms must generate more candidates in the mining process when the minimum share value is small. Moreover, the runtime of the new solution is smaller than MCShFI in a batch way. This is because MCShFI needs to update information on its data structure to meet updated transactions. After that, it has to mine the last updated database to get the final share frequent itemset. Therefore, this process must be performed every time whenever transactions are inserted. For the new solution, it takes advantages of the earlier result of share frequent itemset and does not re-mine the whole updated database. The proposed method mines only the

newly inserted transactions and rescans as necessary to re-calculate itemset from the original database only if it is in Case 3 as mentioned in Section 3.4.

In the next experiments, the minimum share threshold was set at 0.01% and 0.03%, respectively. Both the PSTable and the Frequency table are firstly constructed with 90,000 transactions from T10I4D100K, then, share frequent itemset of the original database is mined from that constructed PSTable. After that, each 2,000 transactions, which were used as new inserted transactions, were performed again with the same thresholds. The results are shown in Figure 4.5 and Figure 4.6.

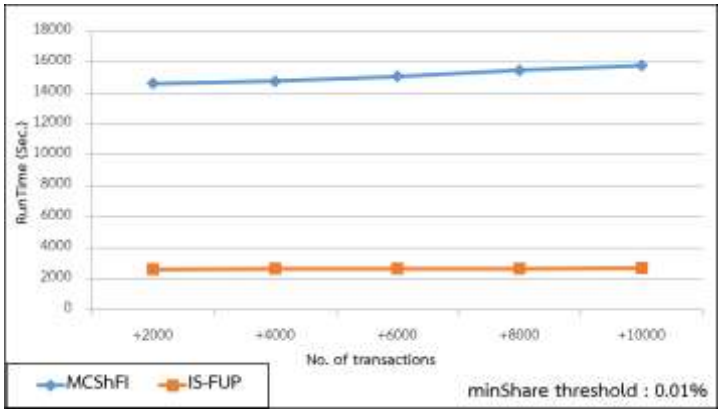


Figure 4.5 The comparison of the execution time for sequentially inserting transactions on T10I4D100K with minShare 0.01%

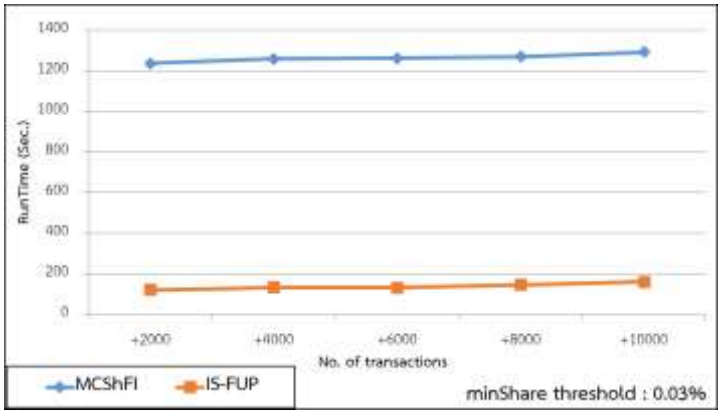


Figure 4.6 The comparison of the execution time for sequentially inserting transactions on T10I4D100K with minShare 0.03%

Similar to above experiments for T40I10D100K, the minimum share threshold was set at 0.1% and 0.3%, respectively. The experimental results are illustrated in Figure 4.7 and Figure 4.8.

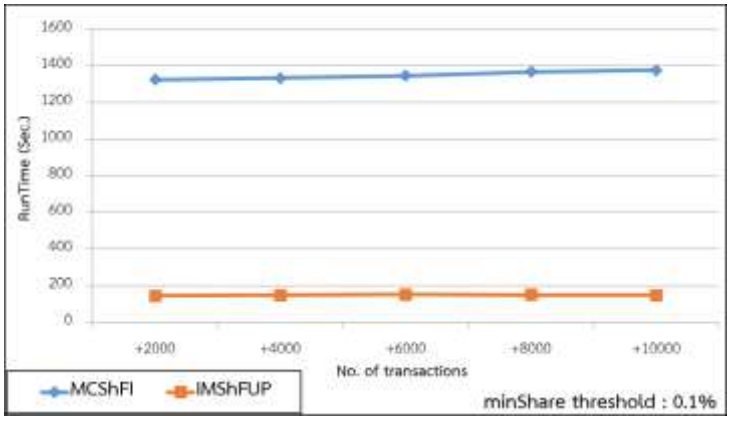


Figure 4.7 The comparison of the execution time for sequentially inserting transactions on T40I10D100K with minShare 0.1%

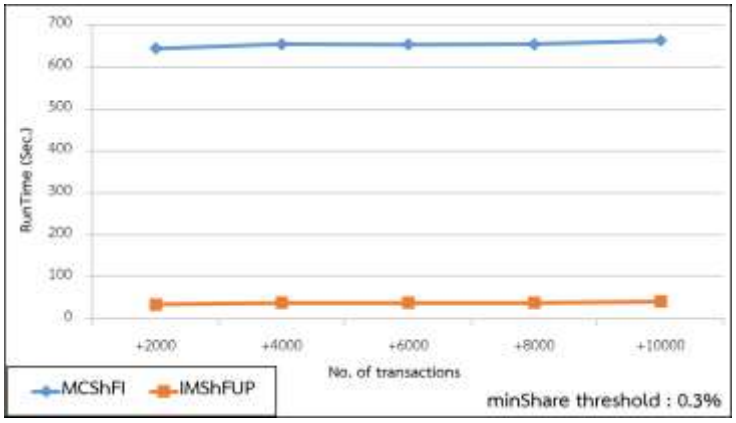


Figure 4.8 The comparison of the execution time for sequentially inserting transactions on T40I10D100K with minShare 0.3%

Figure 4.9 is then made to show the runtimes of the two algorithms on the T10I4D100K dataset for different minimum share threshold. The thresholds range of 0.005 to 0.05 percent were used here. Both the PSTable and the Frequency table are firstly constructed with 90,000 transactions from T10I4D100K, then, share frequent itemset of the original database is mined from that constructed PSTable. After that,

the next 10,000 transactions, which were used as new inserted transactions, were generated with the same thresholds. In the similar way for T40I10D100K, the minimum share thresholds were set from 0.03% to 0.3%. The experimental results illustrated in Figure 4.10.

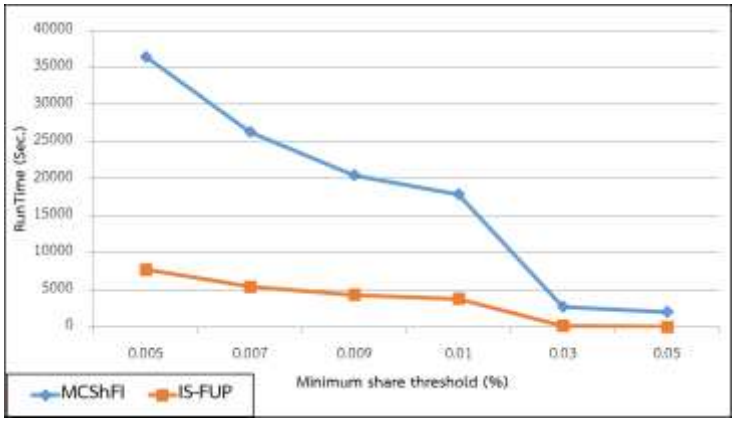


Figure 4.9 The comparisons of the execution time in various minimum share thresholds for T10I4D100K

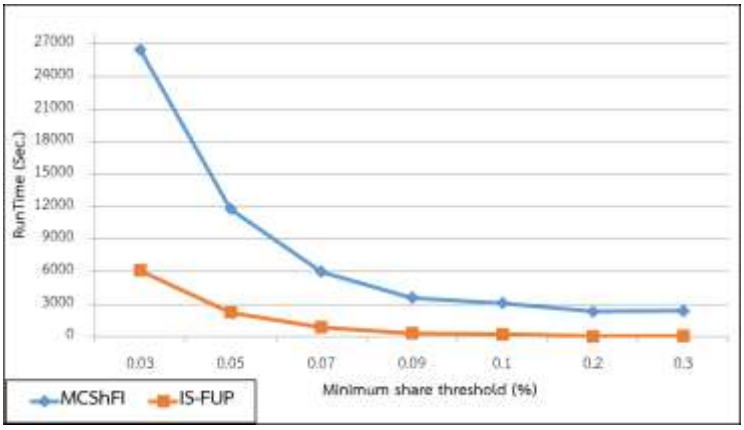


Figure 4.10 The comparisons of the execution time in various minimum share thresholds for T40I10D100K

4.2.2 Experimental results on real-life dataset

In the next experiments, the thresholds were set at 0.5% and 0.6%, respectively. Both the PSTable and the Frequency table are firstly constructed with 34,046 transactions from pumsb*. Then, share frequent itemset of the original database is mined from that constructed PSTable. In addition, various numbers of

transactions of 1,000, 2,000, 3,000, 4,000 and 5,000 were added and the results are shown in Figure 4.11 and Figure 4.12. As shown in these figures, batch MCShFI wasted a lot of time to mine all the whole updated transactions whenever transactions were inserted. The new solution does not re-mine the last updated database and; hence, takes advantages of the previous results of share frequent itemset.

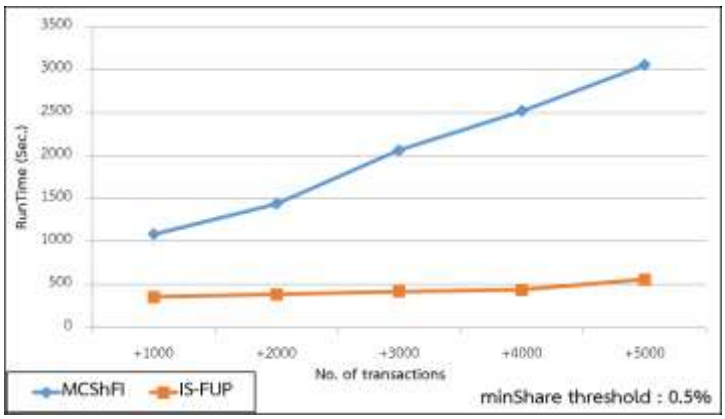


Figure 4.11 The comparison of execution time for various numbers of inserted transactions on pumsb* with minShare 0.5%

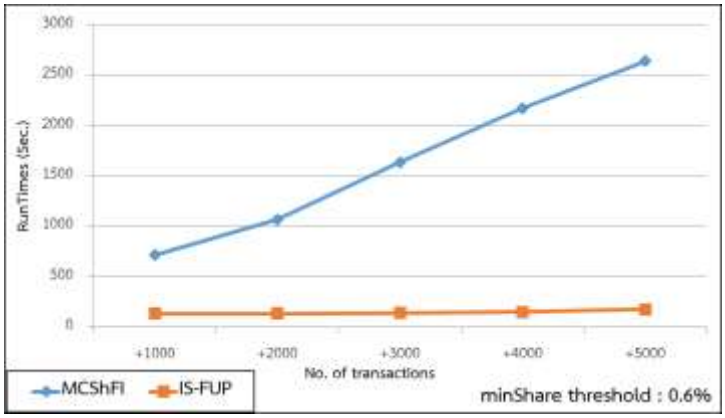


Figure 4.12 The comparison of execution time for various numbers of inserted transactions on pumsb* with minShare 0.6%

In the next experiments, the minimum share threshold was set at 0.5% and 0.6%, respectively. Both the PSTable and the Frequency table are firstly constructed with 44,046 transactions from pumsb*. Then, share frequent itemset of the original database is searched from that constructed PSTable. After that, each 1,000

transactions, which were used as new inserted transactions were generated again with the same threshold. The results are shown in Figure 4.13 and Figure 4.14.

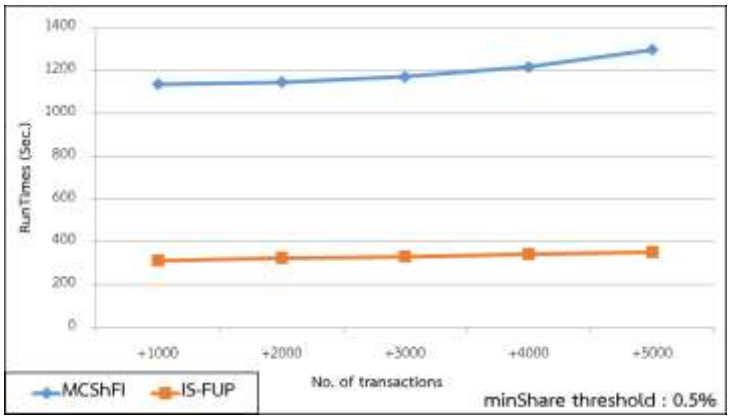


Figure 4.13 The comparison of the execution time for sequentially inserting transactions on pumsb* with minShare 0.5%

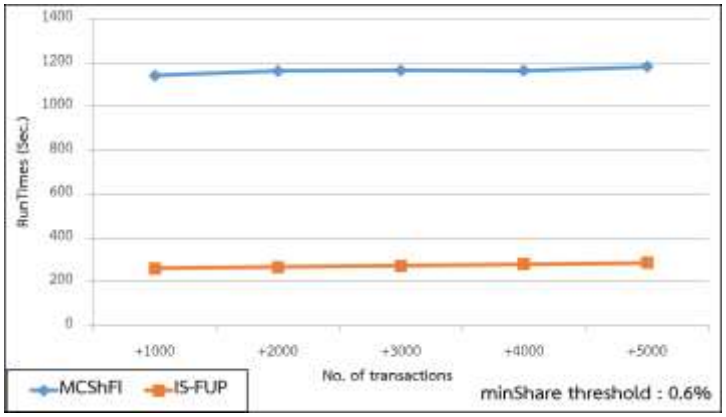


Figure 4.14 The comparison of the execution time for sequentially inserting transactions on pumsb* with minShare 0.6%

The last experiment was then conducted to compare the runtimes of both algorithms on the pumsb* real-life dataset for different numbers of minimum share threshold. Additionally, the minimum share threshold was varied from 0.45% to 0.6%. Both the PSTable and the Frequency table are firstly constructed with 44,046 transactions from pumsb*. Then, share frequent itemset of the original database is searched from that constructed PSTable. After that, the next 5,000 transactions were added, and with the same threshold value, the runtime was measured. The result is as shown in Figure 4.15. The computation time of the new solution is lower than

MCShFI in batch mode for handling transaction insertion. The reason is as same as already mentioned in the previous experiment. In conclusion, the proposed algorithm in incremental share frequent itemset mining (the IS-FUP algorithm) outperforms batch MCShFI to maintain the updated database.

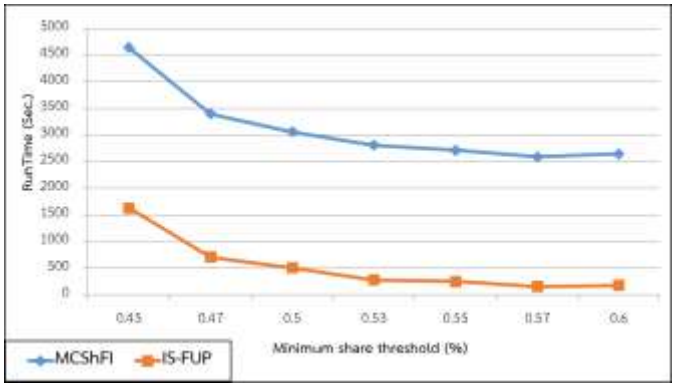


Figure 4.15 The comparisons of the execution time in different minimum share threshold for pumsb*

4.3 Analysis the size of PSTable and Frequency table

In this section, the sizes of the PSTable and Frequency table are analyzed. First, a set of related notations is given as follows.

Let m = the number of all transactions in a database,

= the size of the Frequency table,

n = the number of distinct items in a database,

t = the number of distinct patterns of m transactions,

= the size of the PSTable,

p = the number of transactions sharing same patterns with some other transactions previously recorded.

Since the PSTable is a set of distinct patterns, in other words, it stores only transactions not having any same patterns, therefore, the size of the PSTable is $t = m - p$. Also, because there are n distinct items in the database, by the probability theory, the number of all possible patterns is 2^n. However, to qualify for a legitimate transaction, the empty subset is then excluded from 2^n patterns, that is the maximum number of distinct patterns is 2^n-1. Thus, $t \leq 2^n$-1.

For the size of the Frequency table, since this table records all the transactions without considering duplicate patterns, the size of the PSTable will never exceed the size of the Frequency table, that is, $t \leq m$.

For illustration purposes, the experimental datasets used in this research, as shown in Table 4.2, are examined numerically. In dataset T10I4D100K, m=100,000, n=870 and t=100,000 (since all the transactions in the dataset considered are not include one same pattern at all, size(PSTable) = size(Frequency table)), so 2^{870}-1 = 7.8722E+261 = a relatively large number. Consequently, $t \leq 2^n$-1 and $t \leq m$ holds true as analyzed above. Similar conclusions can be obtained from the other two datasets in Table 4.2.

Table 4.2 Characteristics of three datasets with their size of the PSTable and the Frequency table

Dataset	***m***	***n***	***t***
T10I4D100K	100,000	870	100,000
T40I10D100K	100,000	942	100,000
Pumsb*	49,046	2,088	49,046

Chapter 5
Conclusions and Limitations

5.1 Conclusions

Share Frequent Itemset Mining becomes an important topic in the mining of association rules because it can provide useful knowledge such as total quantity of items sold and total profit. However, the existing methods doesn't support incremental database and need to re-mine the whole updated database to meet the results. This thesis presents an incremental algorithm for mining share frequent itemset, namely IS-FUP, for short, to improve mining execution time with the fast update concept. Among the four updated cases under consideration of the fast update concept, Section 3.6 mathematically shows that the two cases being Case 1 and Case 4 do not affect mining result. This helps in improving execution time of the proposed algorithm. In addition, a new data structure, namely PSTable, that captures all atomic itemset with their count information is presented. The PSTable is constructed once by a single scan database. When new transactions come in, they can suddenly add to the existing database without reconstructing. Extensive performance analyses show that execution time of the incremental algorithm outperforms of the batch one on both synthetic datasets and real-life dataset.

5.2 Limitations

Since the proposed method, IS-FUP, adopts the candidate generate-and-test approach, it requires generating many candidate itemsets; hence, it may still take some time. One possible way in resolving this issue is using the Frequent Pattern Tree model [8] for mining the set of share frequent itemsets without having to generate unnecessary candidate itemsets.

Another possible research idea is to find a way to deal with the time spent on accessing the original database in order to re-calculate the candidate itemsets. This occurs when the candidate itemsets are small in the original database and large in the newly inserted transactions; in other words, it is Case 3 of Section 3.4.

Last but not least, another kind of algorithms incorporating the concept of transaction modification and/or deletion could be developed so as to be applicable to the real-world situation.

References

[1] R. Agarwal, T. Imielinski and A. Swami, "Mining Association Rules Between Sets of Items in Large Database," in *Proceedings of the ACM SIGMOD on Management of Data*, 1993.

[2] R. Agarwal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of 20th International Conference on Very Large Data Bases*, 1994.

[3] D. W. Cheung, J. Han, V. Ng and C. Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," in *Proceedings of International Conference on Data Engineering*, 1996.

[4] C. L. Carter, H. J. Hamilton and N. Cercone, "Share based measures for itemsets," in *Proceedings of the First European Conference on the Principles of Data Mining and Know ledge Discovery*, 1997.

[5] J. S. Park, M. S. Chen and P. S. Yu, "Using A Hash-Based Method with Transaction Trimming for Mining Association Rules," *IEEE Transactions on Knowledge and Data Engineering,* vol. 9, pp. 813-825, 1997.

[6] C. Cai, A. Fu, C. Cheng and W. Kwong, "Mining association rules with weighted items," in *Proceedings of IEEE International Database Engineering and Applications Symposium*, Cardiff, United kingdom, 1998.

[7] B. Barber and H. J. Hamilton, "Algorithms for mining share frequent itemsets containing infrequent subsets," in *Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery*, 2000.

[8] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," Data Mining and Knowledge Discovery, vol. 8, pp. 53-87, 2000.

[9] B. Barber and H. J. Hamilton, "Parametric algorithm for mining share frequent itemsets," *Journal of Intelligent Information Systems,* vol. 16, pp. 277-293, 2001.

[10] J. Pei, J. Han and H. Lu, "Hmine: Hyper-structure mining of frequent patterns in large database," in *Proceedings of International Conference on Data Mining*, 2001.

[11] R. Agarwal, C. Aggarwal and V. V. V. Prasad, "A tree projection algorithm for generation of frequent itemsets," *Journal of Parallel and Distributed,* vol. 61, pp. 350-371, 2001.

[12] B. Barber and H. J. Hamilton, "Extracting share frequent itemsets with infrequent subsets," *Data Mining and Knowledge Discovery,* vol. 7, pp. 153-185, 2003.

[13] R. Chan, Q. Yang and Y. Shen, "Mining high utility itemsets," in *Proceedings of the 3rd IEEE International Conference on Data Mining*, Florida, USA, 2003.

[14] Y. C. Li, J. S. Yeh and C. C. Chang, "Efficient algorithms for mining share-frequent itemsets," in *Proceedings of World Congress of International Fuzzy Systems Association*, 2005.

[15] Y. C. Li, J. S. Yeh and C. C. Chang, "Direct candidates generation: A novel algorithm for discovering complete share-frequent itemsets," in *Proceedings of Fuzzy Systems and Knowledge Discovery*, 2005.

[16] J. Dong and M. Han, " BitTableFI: An efficient mining frequent itemsets algorithm," *Knowledge Based Systems,* vol. 20, pp. 329-335, 2007.

[17] A. Omari , S. Conrad and S. Alcic, "Designing a Well-Structured E-Shop Using Association Rule Mining," in *Proceedings of International Conference on Innovations in Information Technology*, 2007.

[18] T. P. Hong, C. W. Lin and Y. L. Wu, "Incrementally fast updated frequent pattern trees," *Expert Systems with Applications,* vol. 34, pp. 2424-2435, 2008.

[19] P. Gong, C. Yang, H. Li and W. Kou, "The Application of Improved Association Rules Data Mining Algorithm Apriori in CRM," in *Proceedings of International Conference on Pervasive Computing and Applications*, 2007.

[20] S. Liao, C. Hsieh and S. Huang, "Mining product maps for new product development," *Expert Systems with Applications,* vol. 34, pp. 50-62, 2008.

[21] H. Jin, "A Counting Mining Algorithm of Maximum Frequent Itemset Based on Matrix," in *Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery*, 2010.

[22] K. Neelu, A. Neeru and P. K. R., "An Algorithm for Mining Multidimensional Association Rules Using Boolean Matrix," in *Proceedings of International Conference*

on Recent Trends in Information, Telecommunication and Computing, 2010.

[23] N. Khare, N. Adlakha and K. R. Paradasani, "An Algorithm for Mining Multidimensional Association Rules using Boolean Matrix," in *Proceedings of International Conference on Recent Trends in Information, Telecommunication and Computing*, 2010.

[24] C. Nawapornanan and V. Boonjing, "A new share frequent itemsets mining using incremental BitTable knowledge," in *Proceedings of 5th International Conference on Computer Sciences and Convergence Information Technology*, 2011.

[25] C. Nawapornanan and V. Boonjing, "An efficient algorithm for mining complete share-frequent itemsets using BitTable and heuristics," in *Proceedings of International Conference on Machine Learning and Cybernetics*, 2012.

[26] E. Houda, E. F. Mohamed and E. M. Mohammed, "A novel approach for mining frequent itemsets: AprioriMin," in *Proceedings of 4th IEEE International Colloquium on Information Science and Technology*, 2016.

[27] D. P. Shubhangi, R. D. Ratnadeep and K. K. D., "Adaptive Apriori Algorithm for frequent itemset mining," in *Proceedings of International Conference System Modeling & Advancement in Research Trends*, 2016.

[28] D. Aviano , B. L. Putro, E. P. Nugroho and H. Siregar, "Behavioral tracking analysis on learning management system with apriori association rules algorithm," in *Proceedings of International Conference on Science in Information Technology*, 2017.

[29] S. Agrawal and V. Sejwar, "Crime identification using FP-growth and multi objective particle swarm optimization," in *Proceedings of International Conference on Trends in Electronics and Informatics*, 2017.

[30] V. Rajput and A. Manjhvar, "A model for forecasting dengue disease using genetic based weighted FP-growth," in *Proceedings of International Conference on Trends in Electronics and Informatics*, 2017.

[31] X. Zhishuai and L. Wei, "Apriori-based Prediction of the Multi Seat Elections," in *Proceedings of IEEE International Conference on Computational Intelligence and Applications*, 2017.

[32] C. Nawapornanan, S. Intakosum and V. Boonjing, "High Candidates Generation: A New Efficient Method for Mining Share-Frequent Patterns," *Jurnal Teknologi,* vol. 79, pp.

11-19, 2017.

[33] "Frequent Itemset Mining Dataset Repository," [Online]. Available: http://fimi.ua.ac.be/data/. [Accessed 1 August 2018].

Appendices

Appendix A

Journal Publications

Jurnal Teknologi

Full Paper

HIGH CANDIDATES GENERATION: A NEW EFFICIENT METHOD FOR MINING SHARE-FREQUENT PATTERNS

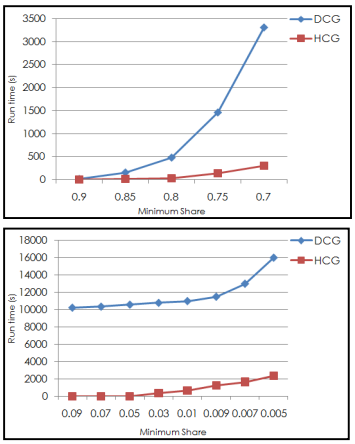
Chayanan Nawapornanan[a*], Sarun Intakosum[a], Veera Boonjing[b]

[a]Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
[b]International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

Article history
Received
28 December 2016
Received in revised form
15 June 2017
Accepted
5 September 2017

*Corresponding author
56605008@kmitl.ac.th

Graphical abstract



Abstract

The share frequent patterns mining is more practical than the traditional frequent patternset mining because it can reflect useful knowledge such as total costs and profits of patterns. Mining share-frequent patterns becomes one of the most important research issue in the data mining. However, previous algorithms extract a large number of candidate and spend a lot of time to generate and test a large number of useless candidate in the mining process. This paper proposes a new efficient method for discovering share-frequent patterns. The new method reduces a number of candidates by generating candidates from only high transaction-measure-value patterns. The downward closure property of transaction-measure-value patterns assures correctness of the proposed method. Experimental results on dense and sparse datasets show that the proposed method is very efficient in terms of execution time. Also, it decreases the number of generated useless candidates in the mining process by at least 70%.

Keywords: Data mining, association rule mining, knowledge discovering, share-frequent patterns mining, frequent patterns mining, frequent itemsets mining

1.0 INTRODUCTION

Frequent patterns mining in a transaction database is an important task for knowledge discovery and data mining such as association rules [1], sequential patterns [3, 4, 21] and classification [14, 19]. Its goal is to find the complete set of patterns that appear with their frequencies in the transaction database above a certain threshold. Numerous methods were proposed to find frequent patterns such as level-wise algorithms [2, 6, 7, 20, 22] and pattern-growth approaches [9, 11, 12, 15]. These methods treated all patterns in a transaction database as a binary (0/1) value. That is, only considering the number of transactions in the database containing desired patterns. However, there could be a transaction with many units of an item. Therefore, discovering only traditional frequent patterns cannot reflect any other implicit factors, such as total costs and profits [13].

In dealing with this problem, Carter *et al*. [5] proposed a new share-frequent pattern mining, which extension of traditional frequent itemsets mining with consideration of the purchased quantities. Several mining methods were proposed for efficiently discovering share-frequent patterns. The Zero pruning (ZP) and the Zero subset pruning (ZSP) proposed by [8, 13] can find all share-frequent patterns using an exhaustive search method. However, both algorithms only prune the generated candidates which have zero local measure value. Some approaches have been proposed to find share-frequent itemsets with infrequent subsets but they cannot extract the complete set of share-frequent patterns such as SIP [8], CAC [10] and IAB [13]. After that, the Fast Share Measure (ShFSM)

method [16] was introduced to solve the weakness of the existing algorithms by using a level-closure property. However, the downward-closure property cannot be kept in this property and the ShFSM still produces too many candidates in each round.

[17] proposed the Direct Candidates Generation algorithm (DCG) to reduce the number of candidates by generating candidates directly without cutting back and joining steps in each iteration. The DCG developed to maintain the property of downward-closure by employing the transaction measure value of a pattern. The authors demonstrate that DCG method outperforms the previous algorithm both on the number of generated candidates and execution time. However, the DCG still extracts a huge number of candidates and consumes a lot of time to generate and test a big number of unwanted candidates in the mining process. This is because it treats all *k*-patterns (patterns with *k* items) as candidates for generating candidates in the later rounds. However, these *k*-patterns could be either high transaction-measure-value patterns or low transaction-measure-value patterns. According to the downward-closure property of transaction-measure-value patterns, we propose to exclude low transaction-measure-value *k*-patterns for generating candidates without losing any share-frequent patternsets.

The organization of this paper is as follows. The next section gives basic definitions. Proposed solution section describes details of our approach. An illustrative example section clarifies the proposed solution. Experimental results section reports performance evaluation of the proposed algorithm. We finally conclude the paper in conclusion section.

2.0 METHODOLOGY

2.1 Basic Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a finite set of items. A subset X of I is called an itemset or a patternset. Let $DB = \{T_1, T_2, .., T_i, .., T_n\}$ be a transaction database where T_i is a subset of I. Associated with each item in a transaction is its quantity sold in the transaction. Table 1 is an example of transaction database with I = {A, B, C, D, E, F, G, H} and $DB = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$.

Table 1 An example transaction database with quantity information

TID	Transaction
T_1	{A:1, B:1, C:1, D:1, G:1, H:1}
T_2	{F:4, H:3}
T_3	{B:4, C:3, D:3}
T_4	{C:4, E:1}
T_5	{B:3, D:2}
T_6	{B:3, C:2, D:1}
T_7	{B:3, C:4, D:1, E:2}
T_8	{A:4, F:1, G:1}
T_9	{C:2, E:1}

Definition 1. The measure value of an item i_p in transaction T_q is the quantity of item i_p in T_q denoted by,

$$mv(i_p, T_q) \tag{1}$$

The measure value of item C in T_1 of Table 1 is $mv(C,T_1) = 1$ which is its associated quantity sold.

Definition 2. The patternset measure value of a patternset X in transaction T_q can be defined as follows,

$$imv(X, T_q) = \sum_{i_p \in X} mv(i_p, T_q) \tag{2}$$

The patternset measure value of a patternset {CD} in transaction T_1 of Table 1 is $imv(\{CD\},T_1) = mv(C,T_1) + mv(D,T_1) = 1+1 = 2$.

Definition 3. The local measure value of a patternset X of a database DB is defined as follows,

$$lmv(X) = \sum_{T_q \in DB_X} \sum_{i_p \in X} imv(i_p, T_q) \tag{3}$$

where DB_X is the set of transaction database containing patternset X.

The local measure value of a patternset {CD} of Table 1 is $lmv(\{CD\}) = imv(\{CD\},T_1) + imv(\{CD\},T_3) + imv(\{CD\},T_6) + imv(\{CD\},T_7) = 2+6+3+5 = 16$.

Definition 4. The total measure value of database is the total measure value of all items in all transactions of a transaction database DB. It is defined by,

$$TMV(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q) \tag{4}$$

The total measure value of Table 1 database is $TMV(DB) = 58$.

Definition 5. The share value of a patternset X in a database DB is the ratio of the local measure value of X to the transaction measure value of DB. That is,

$$SH(X) = \frac{lmv(X)}{TMV(DB)} \tag{5}$$

The share value of a patternset {CD} in Table 1 database is $SH(\{CD\}) = lmv(\{CD\})/TMV(DB) = 16/58 = 0.27$.

Definition 6. Let δ be a minimum share threshold where $0 \leq \delta \leq 1$. The minimum local measure value of database DB with a threshold δ is defined as follows,

$$\min\_lmv = ceiling(\delta \times TMV(DB)) \tag{6}$$

The minimum local measure value of Table 1 database with $\delta = 0.25$ is $min\_lmv = \text{ceiling}(0.25 \times 58) = 15$.

Definition 7. The transaction measure value of transaction T_q is defined as follows,

$$tmv(T_q) = \sum_{i_p \in T_q} mv(ip, Tq) \quad (7)$$

From Table 1 database, the transaction measure value of T_1 is $tmv(T_1) = 6$.

Definition 8. The transaction measure value of a patternset X of a database DB is defined as follows,

$$tmv(X) = \sum_{X \subseteq T_q \in DB_X} tmv(Tq) \quad (8)$$

where DB_X is the set of transaction database containing patternset X.

The transaction measure value of patternset {F} of Table 1 database is $tmv(\{F\}) = tmv(T_2) + tmv(T_8) = 7 + 6 = 13$.

Lemma 1. The transaction measure value of a patternset X maintains the property of downward closure.

Proof. Let {A} be a share-frequent patternsets patternset in $DB_{\{A\}}$, where $DB_{\{A\}}$ is a set of all transaction database containing {A}. Let {B} be a super pattern of {A}, therefore {B} cannot be present in any transaction that {A} is absent. Thus, refer to definition 8, the maximum transaction measure value of {B} is $tmv(\{A\})$. Therefore, if $tmv(\{A\})$ is less than min_lmv, {B} cannot be a share-frequent patternsets and obviously cannot be a high transaction measure value pattern (including {A}).

By Lemma 1, if a patternset X is not high transaction measure value pattern, its supersets are not high transaction measure value patterns either. From Table 1 database with δ = 0.25, we have $min\_lmv = 15$ and $tmv(\{F\}) = 13 < min\_lmv$. Therefore, all supersets of {F} are not high transaction measure value patterns.

Definition 9. A patternset X of a database DB is a high transaction measure value patternset with respect to a threshold δ, if $tmv(X) \geq min\_lmv$.

2.2 Proposed Solution

In this section, at first, we present an efficient data structure called "a PSTable Knowledge" to maintain an incremental database. After that, the HCG algorithm is presented for mining share-frequent patterns from the PSTable Knowledge and it generates the candidate patterns from only the high transaction measure value patterns.

2.2.1 Maintenance of PSTable Knowledge

The PSTable is a set of patterns with their quantities. It is an improved version of a BitTable structure [18] by aggregating transactions having similar patterns. The maintaining process of the PSTable is described as below.

To maintain PSTable, each transaction from initial database or the newly inserted transaction is loaded and treated as a patternset. After that, it checks the patternset of the transaction whether it appears in the PSTable or not. If it does, the quantity of the transaction (total count) is summed with the new entries to gain new value of total count, and the patternset count is incremented by 1. Otherwise, it is the new patternset and then it will be inserted into the PSTable with the quantity of the transaction and set the patternset count to be 1. All the mentioned steps are repeated for all of the transaction in initial database or the newly inserted transactions. The pseudocode of the algorithm is shown in Algorithm 1 in Figure 1.

```
Algorithm 1 Maintenance the PSTable Knowledge
Input : The PSTable, group of original transaction database (DB) or
the newly inserted transactions (db+)
Output : The updated PSTable
1. For each transaction t in DB or db+
2.      set X be a patternset of t
3.      If X is not in PSTable then
4.          add X to the PSTable
5.          set a Total Count as the total of quantity of X
6.          set a Patternset Count as 1
7.      Else
8.          add total quantity of X to a Total Count
9.          increase a Patternset Count by 1
10.     End If
11.End For
```

Figure 1 Pseudocode for maintaining PSTable Knowledge

Example 1. To illustrate the maintaining process of the PSTable, we use a database shown in Table 1 as an example.

The first transaction T_1 ={A:1, B:1, C:1, D:1, G:1, H:1} is loaded into the algorithm, that is, {A,B,C,D,G,H} is treated as a new patternset. Next, the new patternset is checked in the PSTable. The new patternset is new, it will be added to the PSTable and set the total count as the sum of the quantity of transaction (=6) and also set the value 1 to the patternset count. The algorithm does the same steps to T_5 and the result is illustrated in Figure 2 (a). After that the next transaction T_6 = {B:3, C:2, D:1} is read to the algorithm, the new patternset of T_6 is {B,C,D} and it is examined in the PSTable. The new patternset exists, the total count is concluded with the new entries to gain new value of total count as (10+6 = 16) and also increases by one to the patternset count (1+1 = 2) as shown in Figure 2(b). This process is repeated for all transactions and the final PSTable Knowledge is presented in Figure 2(c).

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					10	1
P_4			X		X				5	1
P_5		X		X					5	1

(a) The result of maintaining algorithm after transaction T1-T5 is added

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					16	2
P_4			X		X				5	1
P_5		X		X					5	1

(b) The result of maintaining algorithm after transaction T6 is added

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					16	2
P_4			X		X				8	2
P_5		X		X					5	1
P_6		X	X	X	X				10	1
P_7	X					X	X		6	1

(c) The result of the maintaining algorithm

Figure 2 PSTable knowledge extracted from Table 1 database

2.2.2 The HCG Algorithm

In this subsection, we present a new efficient algorithm for mining share-frequent patterns from the PSTable Knowledge. It generates the minimized number of candidate from only high transaction measure value patternsets. As mentioned in Lemma 1, the transaction measure value (tmv) contains the downward closure property, therefore, it is used to cutting back the useless candidate patternsets in each iteration. To determine whether the patternset is high or not, the value of *tmv* is calculated (by definition 8) - if the value of *tmv* of any patternsets is larger than or equal to the minimum local measure value (*min_lmv*), it is in the set of high transaction measure value. The details of the HCG algorithm are described below.

Input:
1. The PSTable Knowledge
2. A minimum share threshold δ

Output: High share-frequent patterns, C

Step 1: Calculates the total measure value of database from the PSTable using Eq. (4) denoted by *TMV*. After that, the *min_lmv* is computed by using Eq. (6).

Step 2: Each *1*-patternset from the PSTable *X* is loaded into the HCG algorithm and then calculates the transaction measure value of *X* using Eq. (8) denoted by *tmv*(*X*).

Step 3: Checks whether the value of *tmv*(*X*) is larger than or equal to the *min_lmv*. If *X* satisfies the above condition, put *X* in a set of high transaction measure value (abbreviated as *Htmv*).

Lemma 2. Each *1*-patternset from the PSTable *X* is an infrequent if *tmv*(*X*) <*min_lmv*

Proof. Let *X* be a high transaction measure value in the PSTable. According to definition 8, *tmv*(*X*) must be greater than or equal to *min_lmv*. Therefore, this opposes to the assumption *tmv*(*X*)<*min_lmv*. Thus, a patternset *X* is an infrequent.

In HCG algorithm, we calculate all *Htmv* in the PSTable by determining the original transaction measure value according to Eq. (8) which this value maintains the property of downward closure.

Step 4: Each *1*-patternset in the high transaction measure value (*Htmv*) *X* is read into the HCG algorithm. Then, the algorithm will show every pattern from the $PSTable_X$ where $PSTable_X$ is the set of patterns containing patternset *X*. The transaction measure value of each *1*-patternset in $PSTable_X$ is computed.

Step 5: Checks whether the value of *tmv* of each *1*-patternset in $PSTable_X$ is greater than or equal to the *min_lmv*. If *X* satisfies the above condition, put *X* in a set of high transaction measure value of *X*, $Htmv_X$.

Lemma 3. Each *1*-patternset from the $PSTable_X$ *Y* is an infrequent if *tmv*(*Y*) <*min_lmv*

Proof. Let *Y* be a high transaction measure value in the $PSTable_X$. According to definition 8, *tmv*(*Y*) must be greater than or equal to *min_lmv*. Therefore, this opposes to the assumption of *tmv*(*Y*) <*min_lmv*. Thus, a patternset *Y* is an infrequent.

In our algorithm, we calculate all $Htmv_X$ in the $PSTable_X$ by performing the original transaction measure value calculation according to Eq. (8) which this value maintains the property of downward closure.

Step 6: Mines all 2-patternset in $Htmv_X$ that prefixs with *X* and keep them in the C_2. After that, removes *X* from *Htmv* and sets *k* = 3, where *k* is used for recording the length of generated candidates in the mining process.

Step 6-1: The candidate *k*-patternsets are generated from C_{k-1} and also compute their transaction measure value (*tmv*(*k*-patternsets)) in the $PSTable_X$ for

checking against the *min_lmv*. Checks whether the value of *tmv*(*k*-patternset) is greater than or equal to the *min_lmv*. If it satisfies the above condition, stores it to C_k. Increases *k* by one and this step is then repeated until no candidate patternset is generated.

Step 7: Repeats Steps 4 – 6 until the *Htmv* is empty and returns the *C* that stores all high transaction-measure-value pattern at the end of the HCG algorithm.

2.3 An Illustrative Example

In this section, we use the Table 1 database to illustrate the HCG algorithm step by step. Also, it is assumed that the minimum share threshold is 25% of total quantity.

Input:
1. The PSTable is shown in Figure 2 (c), which constructed from the example transaction database in Table 1.
2. The minimum share threshold (δ) is set as 0.25.

Output: High share-frequent patterns, *C*

Step 1: The HCG algorithm calculates the initial variables at first i.e. *TMV*(=58) and *min_lmv* (58*0.25), which is 15, respectively.

Step 2: The *1*-patternset *X* in the PSTable for {A}, {B}, {C}, {D}, {E}, {F}, {G} and {H} are read into the algorithm and then calculates their transaction measure value (*tmv*(*X*)). Takes *1*-patternset {A} as an example to illustrate the process. The *1*-patternset {A} appears in the PSTable P_1 and P_7 and then the transaction measure value of the *1*-patternset {A} is calculated as (6+6 = 12). The other *1*-itemsets are calculated in the same way and the result is shown in Figure 3.

1-Patternset *X*	*tmv*(*X*)
A	12
B	37
C	40
D	37
E	18
F	13
G	12
H	13

Figure 3 *tmv*s of *1*-patternsets

Step 3: The *tmv* values of the *1*-patternset are checked against the *min_lmv*. In this example, the four item {B}, {C}, {D} and {E} satisfy the condition and put them in both the set of high transaction measure value (*Htmv*) and C_1. Thus, *Htmv* = {{B}, {C}, {D}, {E}}.

Step 4: Each *1*-patternset *X* in the *Htmv* is then read into the algorithm. Takes $Htmv_B$ as an example, the {B} appears in Pattern P_1, P_3, P_5 and P_6 in the PSTable ($PStable_B$). Then, the HCG algorithm computes the *tmv* value of all *1*-patternset in $PStable_B$ as {{B: 6+16+5+10 (=37)}, {C: 6+16+10 (=32)}, {D: 6+16+5+10 (=37)} and {E: 10}}, respectively, is shown in Figure 4 (a).

Step 5: Checks whether the *tmv* value of the *1*-patternset in the $PStable_X$ is larger than or equal to the *min_lmv*. Then, put them in the set of high transaction measure value of *X*, $Htmv_X$. In this example, the *1*-patternsets in the $PStable_B$ satisfy the condition which are as follows {B}, {C} and {D}. Thus, $Htmv_B$ = {B,C,D}. The other $Htmv_X$ are then processed in the same way. The results are shown in Figure 4 (b)-(d) respectively.

PID	B	C	D	E	TotalCount
P_1	X	X	X		6
P_3	X	X	X		16
P_5	X		X		5
P_6	X	X	X	X	10
	37	**32**	**37**	**10**	

(a) The process of {B} in Htmv

PID	C	D	E	TotalCount
P_1	X	X		6
P_3	X	X		16
P_4	X		X	8
P_7	X	X	X	10
	40	**32**	**18**	

(b) The process of {C} in Htmv

PID	D	E	TotalCount
P_1	X		6
P_3	X		16
P_5	X		5
P_6	X	X	10
	37	**10**	

(c) The process of {D} in Htmv

PID	E	TotalCount
P_4	X	8
P_6	X	10
	18	

(d) The process of {E} in Htmv

Figure 4 The process of 1-patterns in *Htmv*

Step 6: Mines all 2-patternset in $Htmv_X$ that prefix with *X* and then removes *X* from *Htmv*. Takes $Htmv_B$ as an example to illustrate the process. The HCG generates candidate 2-patternset of the $Htmv_B$ as {B,C} and

{B,D} respectively, which is shown in Figure 5, and also keeps them in C_2. Then, *k* is set to 3.

Step 6-1: The candidate *k*-patternset are generated from C_{k-1} and also compute their transaction measure value for checking against the *min_lmv*. From the result of candidate 2-patternset of the $Htmv_B$, the HCG generates the candidate *k*-patternset which is {B,C,D} as shown in Figure 6 and it satisfies the condition, saves it to C_k. This sub step of pattern "B" can be terminated in the third round because no candidate patternset is generated.

2-Patternset *X*	*tmv*(*X*)
{BC}	32
{BD}	37

Figure 5 *tmv*s of 2-patternsets with prefix "B"

3-Patternset *X*	*tmv*(*X*)
{BCD}	32

Figure 6 *tmv*s of 3-patternsets with prefix "B"

Step 7: Steps 4 – 6 are then repeated until no member of *Htmv*. After the algorithm works complete, we get *C* = {{B}, {C}, {D}, {E}, {B,C}, {B,D}, {B,C,D}, {C,D}, {C,E}}. The share-frequent patterns mined from *C* are {C}, {B,C}, {B,D}, {C,D} and {B,C,D} respectively. All generated candidate are shown in Figure 7.

No.	Candidate Patterns	tmv(*X*)	lmv(*X*)	Sh(*X*)	Share-frequent patterns
1	B	37	14	0.2413	NO
2	C	40	18	0.2758	YES
3	D	37	7	0.1379	NO
4	E	18	5	0.0689	NO
5	BC	32	21	0.3620	YES
6	BD	37	22	0.3793	YES
7	BCD	32	27	0.4655	YES
8	CD	32	16	0.2758	YES
9	CE	18	14	0.2413	NO

Figure 7 The generated candidates by the HCG algorithm

Observation 1. The DCG algorithm adopts the level-wise candidate generation-and-test methodology. Firstly, it reads all *1*-patternset from a transaction database and treats them as *1*-candidate for generating all the candidate for length 2, that is, all *1*-candidate contain high *tmv* value and low *tmv* value. For example, if the number of atomic patterns is 500 and they are treated as *1*-candidate and then the algorithm tests for $\binom{500}{2}$ 2-patternset candidate patterns. For 2-patternset, a patternset will be considered for selection as a candidate by the definition 8. Therefore, a large number of candidates are generated.

Lemma 4. If M_1 is the number of generated candidates by HCG algorithm and M_2 is the number of generated candidates by DCG algorithm, then $M_1 \subseteq M_2$.

Proof. Let $Z\{z_1,z_2, \ldots ,z_n\}$ be a candidate patternset if all of its subset of length *n*-1 are candidate patternset (high transaction measure value) in the DCG algorithm. So, *Z* could have low *tmv* value to become a candidate of length *1*. In HCG algorithm, if *Z* is a low *tmv* then it cannot appear as a candidate. In addition, HCG prunes *Z* immediately after determining it is low *tmv* value. In conclusion, the generated candidate patternsets of HCG stores only the high transaction measure value, therefore, M_1 is less than or equals to M_2.

3.0 RESULTS AND DISCUSSION

3.1 Experimental Environment and Datasets

In this section, the experimental evaluation of the compared algorithms: the HCG solution and the DCG method [17] are conducted. All algorithms were implemented in Microsoft C# 2012 and running on a 2.60 GHz Intel(R) Core i7-4720HQ with 12 GB main memory on the Windows 8.1 operating system. Similar to the performance evaluation of the previous share-frequent patterns mining algorithm [17], the count information was assigned to each item in each transaction, ranging from 1 to 10. Four datasets are used in the experiments and they are acquired from FIMI Repository Page [23-26]. The characteristics of the datasets are shown in Table 2, |D| is the total number of transaction in a dataset, |I| is the number of distinct pattern in the dataset and T_{avg} is the average size of transaction. The dataset is dense since the number of atomic pattern is small and each transaction has a lot of distinct pattern. By considering the mushroom dataset in Table 2, its atomic pattern is 119 and its transaction size is 23. Therefore, the ratio of its distinct pattern presented in every transaction is 20% ((23/119)*100).

Table 2 Characteristics of the experiment datasets

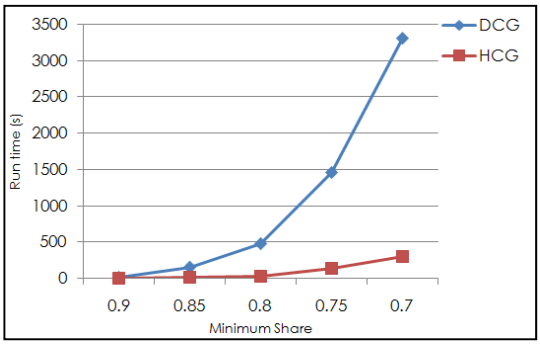
Dataset	|D|	|I|	T_{avg}	Type
Mushroom	8,124	119	23	Dense
Chess	3,196	75	37	Dense
T10I4D100K	100,000	870	10.1	Sparse
T40I10D100K	100,000	942	40.5	Sparse

3.2 Experimental Results for Dense Datasets

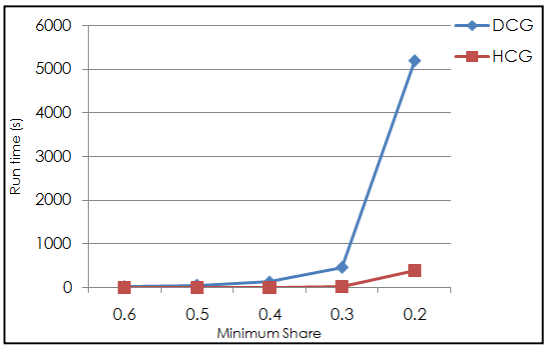
In the first experiment, we compare the running time of the HCG algorithm over the DCG method using the Mushroom and Chess datasets. As there is no share-frequent patternset, two ranges of minShare are chosen. For Mushroom dataset, the minimum share threshold is set from 0.6 to 0.2, while for Chess

dataset, it is set from 0.9 to 0.7. The results are shown in Figure 8 (a) and (b). It is obvious to see that two algorithms require more runtime when the minimum share threshold decreases. This is reasonable because when the threshold value became smaller, more candidates in the mining process are generated.

Moreover, the HCG algorithm performs better than the existing DCG algorithm with respect to all of the minimum share thresholds. The reason is that DCG requires more execution time to generate and test a huge number of useless candidates in the mining process since it treated all *1*-patterns as candidates for generating candidates in the later rounds. That is, *1*-patterns contain both high *tmv* patterns and low *tmv* patterns. For the HCG algorithm, it generates the candidate from only high transaction measure value atomic patterns in the second round. For the third round onwards, it created candidates from the combination of only the patterns which have their transaction measure value beyond a certain threshold. The number of candidate in the mining process and the number of generated candidate of both methods are shown in Table 3 and 4 respectively.



(a) Total running time on Chess



(b) Total running time on Mushroom

Figure 8 Running times of DCG and HCG with different thresholds on dense datasets

Table 3 The number of generated candidates of DCG and HCG on the chess dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.6	8,623	179	162	51
0.5	12,003	331	259	153
0.4	25,296	967	663	565
0.3	90,924	3,744	2,826	2,735
0.2	950,253	58,760	53,697	53,621

Table 4 The number of generated candidates of DCG and HCG on the mushroom dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.90	12,542	955	691	629
0.85	41,797	3,487	2,733	2,674
0.80	116,920	10,256	8,304	8,248
0.75	282,912	25,958	21,018	20,966
0.70	629,379	60,377	48,972	48,921

Table 5 compares the number of candidates in the mining process and the number of generated candidates of two algorithms in each round. A Mushroom dataset was employed and the minimum share threshold was set at 0.3%. In addition, we further found that, in every round of mining process, the HCG method produces a smaller number of candidates than that of the DCG one. The reason to this situation is the same as the content mentioned earlier. The HCG and the DCG methods terminate in the ninth round and tenth round respectively.

Table 5 The number of candidates in the mining process and the number of generated candidates on Mushroom dataset

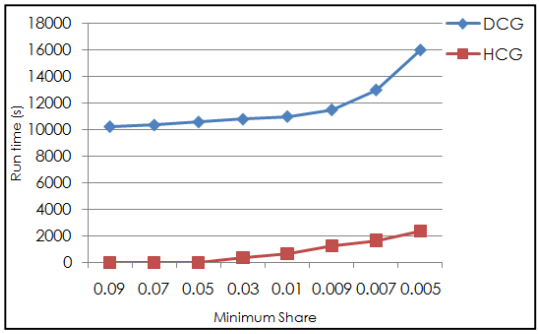
k-patternset	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
k=1	119	119	119	28
k=2	7,021	378	163	163
k=3	7,164	660	455	455
k=4	15,836	956	725	725
k=5	22,179	856	712	712
k=6	20,482	525	441	441
k=7	12,315	203	169	169
k=8	4,653	43	38	38
k=9	1,044	4	4	4
k=10	111	-	0	-
Total	90,924	3,744	2,826	2,735

3.3 Experimental Results for Sparse Datasets

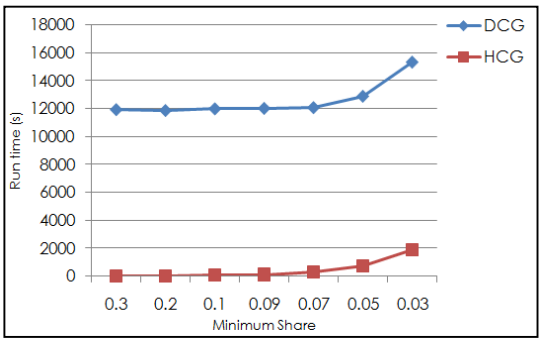
In this experiment, we present the comparison of HCG and DCG algorithms on sparse datasets,

T10I4D100K and T40I10D100K, the results in terms of consumed run time are shown in Figure 9 (a) and (b). Two ranges of min_share are chosen. For T10I4D100K dataset, the minimum share threshold was set from 0.09 to 0.005. While for T40I4D100K dataset, it was set from 0.3 to 0.05, as there is no share-frequent patternset.

It shows that our proposed HCG runs faster than the DCG one since the new proposed HCG can reduce the large number of unpromising candidates in the mining process because the HCG generated the candidate patterns from only the patterns with transaction measure value above a certain threshold.



(a) Total running time on T10I10D100K



(b) Total running time on T40I4D100K

Figure 9 Running times of DCG and HCG with different thresholds on sparse datasets

Table 6 and 7 compares the number of candidates in the mining process and the number of generated candidates of two algorithms. Obviously, from Table 6, the minimum share is set to 0.09, it could be seen that no candidate is larger than a certain threshold. The process of HCG starts to read all *1*-patterns, which is 870 items and then checks whether their values is high and no pattern is above a certain threshold (high), therefore the HCG terminates in the first round. For DCG algorithm, it reads all *1*-patterns and treats them as *1*-candidates (870 items) for generating-and-testing 2-patterns in second round. Then, no pattern for 2-patterns is above a certain threshold, so the DCG terminates in the second round.

Table 6 The number of generated candidates of DCG and HCG on the T10I4D100K dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.30	444,153	942	942	0
0.20	444,153	957	942	6
0.10	444,153	5,037	942	91
0.09	444,153	7,845	942	118
0.07	444,760	20,249	944	199
0.05	452,284	50,491	966	339
0.03	574,350	128,842	1,369	926

Table 7 The number of generated candidates of DCG and HCG on the T40I10D100K dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.090	378,885	870	870	0
0.070	378,885	876	870	4
0.050	378,885	1,023	870	18
0.030	378,885	5,241	870	94
0.010	384,432	85,545	889	431
0.009	397,609	99,694	929	504
0.007	435,750	131,367	1,105	746
0.005	605,839	185,077	1,797	1,532

Similar to Table 5, Table 8 compares the difference of the candidate numbers in the mining process and the generated candidate numbers between both algorithms in each pass. The T10I4D100K dataset was used and *min_share* was set to 0.009. We can observe that the HCG generates lower numbers candidate than that of the DCG method. In addition, the overall performance of HCG is better because it produces the candidate from only high the transaction measure value patterns. Through all of the experimental results, we can observe that the new proposed effectively much better than the existing DCG algorithm in terms of the total running time and the number of generated candidates on both dense and sparse datasets.

Table 8 The number of candidates in the mining process and the number of generated candidates on T40I10D100K dataset

k-patternset	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
k=1	870	870	870	445
k=2	378,015	98,790	50	50
k=3	14,916	33	9	9
k=4	3,808	1	0	0
Total	397,609	99,694	929	504

4.0 CONCLUSION

This paper presented a new efficient method for mining share-frequent patterns, named HCG, aimed to develop for decreasing the number of unpromising candidate patterns. The HCG generated the candidate pattern from only the patterns which have their transaction measure value beyond a certain threshold in a level-wise way. Moreover, we also presented a new data structure that captures all atomic patterns with their count information, named PSTable, which is constructed once by a single scan database. When new transactions come in, they can suddenly add to the existing database without reconstructing. Extensive performance analysis shows that our approach outperforms the existing DCG algorithm in terms of the total running time and the number of generated candidates on both dense and sparse datasets.

References

[1] Agarwal, R., Imielinski, T. and Swami, A. 1993. Mining Association Rules between Sets of Items in Large Database. *Proceedings of the ACM SIGMOD on Management of Data*. 207-216.

[2] Agarwal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of 20th International Conference on Very Large Data Bases*. 487-499.

[3] Agarwal, R. and Srikant, R. 1995. Mining Sequential Patterns. *Proceedings of 11th International Conference on Data Engineering*. 3-14.

[4] Agarwal, R. and Srikant, R. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. *Proceedings of 5th International Conference on Extending Database Technology*. 3-17.

[5] Carter, C. L., Hamilton, H. J. and Cercone, N. 1997. Share Based Measures for Itemsets. *Lecture Notes in Computer Science*. 1263: 14-24.

[6] Park, J. S., Chen, M. S. and Yu, P. S. 1997. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*. 9: 813-825.

[7] Brin, S., Motwani, R., Ullman, J. D. and Tsur, S. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Proceedings of the ACM SIGMOD on Management of Data*. 255-264.

[8] Barber, B. and Hamilton, H. J. 2000. Algorithms for Mining Share Frequent Itemsets Containing Infrequent Subsets. *Lecture Notes in Computer Science*. 1910: 316-324.

[9] Han, J., Pei, J. and Yin, Y. 2000. Mining Frequent Patterns without Candidate Generation. *Proceedings of the ACM SIGMOD on Management of Data*. 1-12.

[10] Barber, B. and Hamilton, H. J. 2001. Parametric Algorithm for Mining Share Frequent Itemsets. *Journal of Intelligent Information Systems*. 16: 277-293.

[11] Pei, J., Han, J. and Lu, H. 2001. Hmine: Hyper-structure Mining of Frequent Patterns in Large Database. *Proceedings of International Conference on Data Mining*. 441-448.

[12] Agarwal, R., Aggarwal, C. and Prasad, V. V. V. 2001. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed*. 61: 350-371.

[13] Barber, B. and Hamilton, H. J. 2003. Extracting Share Frequent Itemsets with Infrequent Subsets. *Data Mining and Knowledge Discovery*. 7: 153-185.

[14] Han, J., Pei, J., Yin, Y. and Shi, C. 2004. Integrating Classification and Association Rule Mining: A Concept Lattice Framework. *Lecture Notes in Computer Science*. 1711: 443-447.

[15] El-Hajj, M. and Zaiane, O. R. 2004. COFI Approach for Mining Frequent Itemsets Revisited. *Proceeding of the ACM SIGMOD on Data Mining and Knowledge Discovery*. 70-75.

[16] Li, Y. C., Yeh, J. S. and Chang, C. C. 2005. A Fast Algorithm for Mining Share-frequent Itemsets. *Lecture Notes in Computer Science*. 3399: 417-428.

[17] Li, Y. C., Yeh, J. S. and Chang, C. C. 2005. Direct Candidates Generation: A Novel Algorithm for Discovering Complete Share-frequent Itemsets. *Lecture Notes in Computer Science*. 3614: 551-560.

[18] Nawapornanan, C and Boonjing, V. 2011. A New Share Frequent Itemsets Mining Using Incremental Bittable Knowledge. *Proceedings of 5th International Conference on Computer Sciences and Convergence Information Technology*. 358-362.

[19] Mohammad, N. Q., Hassan, F. H. A., Yahya, K. T. 2015. An Improved Documents Classification Technique Using Association Rules Mining. *Proceedings of IEEE International Conference on Research in Computational Intelligence and Communication Networks*. 460-465.

[20] Houda, E., Mohamed, E. F. and Mohammed, E. M. 2016. A Novel Approach for Mining Frequent Itemsets: AprioriMin. *Proceedings of 4th IEEE International Colloquium on Information Science and Technology*. 286-289.

[21] Peng, H. 2016. Improved Algorithm Based on Sequential Pattern Mining of Big Data Set. *Proceedings of 7th IEEE International Conference on Software Engineering and Service Science*. 115-118.

[22] Shubhangi, D. P., Ratnadeep, R. D. and D., K. K. 2016. Adaptive Apriori Algorithm for Frequent Itemset Mining. *Proceedings of International Conference System Modeling & Advancement in Research Trends*. 7-13.

[23] Frequent Itemset Mining Dataset Repository, "Chess", http://fimi.ua.ac.be/data/, 1987 (Accessed: June 9, 2017).

[24] Frequent Itemset Mining Dataset Repository, "Mushroom", http://fimi.ua.ac.be/data/, 1989 (Accessed: June 9, 2017

[25] Frequent Itemset Mining Dataset Repository, "T10I4D100K", http://fimi.ua.ac.be/data/, 2003 ((Accessed: June 9, 2017).

[26] Frequent Itemset Mining Dataset Repository, "T40I10D100K", http://fimi.ua.ac.be/data/, 2003 (Accessed: June 9, 2017).

Thai Journal of Mathematics
Volume 16 (2018) Number 1 : 1–23

http://thaijmath.in.cmu.ac.th
ISSN 1686-0209

An Incremental Approach to Share-Frequent Itemsets Mining

Chayanan Nawapornanan[†,1], Sarun Intakosum[†] and Veera Boonjing[‡]

[†]Department of Computer Science, Faculty of Science
King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
e-mail : 56605008@kmitl.ac.th (C. Nawapornanan)
kisarun@kmitl.ac.th (S. Intakosum)

[‡]International College King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
e-mail : kbveera@kmitl.ac.th (V. Boonjing)

Abstract : The share-frequent itemsets mining becomes an important topic in the mining of association rules because it can provide useful knowledge such as total quantity of items sold and total profit. In the past, the efficient MCShFI algorithm was successfully proposed to discover complete share-frequent itemsets on a database. When the database is updated, the algorithm can obtain current complete share-frequent itemsets by using the batch approach-mining the whole updated database. To improve mining execution time, we propose a new incremental approach to the problem with the Fast Update (FUP) concept. It obtains the current result by mining only new transactions and updating the previous existing result with this mined result.

Keywords : data mining; share-frequent itemsets mining; incremental mining.
2010 Mathematics Subject Classification : 68T20; 68W01.

1 Introduction

Mining frequent itemsets is the essential process and most expensive in association rule mining [1]. Many algorithms of mining frequent itemsets have been

[1]Corresponding author.

proposed to find frequent patterns such as level-wise approaches [2, 3, 4, 5, 6] and pattern-growth algorithms [7, 8, 9, 10]. The frequent itemsets mining has some limitations, that is, in each transaction, each item will appear in a binary frequency (either absent or present). Moreover, it only considers whether an item is bought in to a transaction or not [11]. Therefore, mining share-frequent itemsets [12] was proposed to accommodate the quantity of each item. Its share measure can provide useful knowledge about the numerical values that are typically associated with the transaction items, such as the total quantity of items sold or the total profit. [13] proposed an efficient algorithm for mining complete share-frequent itemsets, named MCShFI, for efficiently extracting complete share-frequent itemsets based on a level-wise generating property. It can reduce the runtimes by using transaction measure value as an effective upper bound of each item to generate only the promising candidate itemsets in each iteration. To obtain a current mining result from an updated database, however, this algorithm has to rescan the whole database. This paper proposes an incremental algorithm called IS-FUP which eliminates the rescanning by mining frequent itemsets only from new transactions and using this result to efficiently update the previous discovered frequent itemsets. It adopts the Fast Update (FUP) Algorithm [14] as an efficient update concept.

The remaining of this paper is organized as follows. Background and related work are given in Section 2. The share-frequent itemsets mining problem is described in Section 3. In Section 4, the proposed algorithm is described and some examples are given. The efficiency of IS-FUP is shown in Section 5 and conclusions are drawn in Section 6.

2 Related Work

2.1 Mining Share-Frequent Itemsets

In 1997, Carter et al. introduced the share-confidence model to discover useful knowledge about numerical attributes associated with items in a transaction [12]. This numerical attribute reflects all the profit, the cost of items, and associates to the transaction items. Several mining techniques have been proposed for efficiently discovering share-frequent itemsets. [15] and [11] proposed method named Zero pruning (ZP) and Zero subset pruning (ZSP) respectively that can extract the complete set of share-frequent itemsets from the transaction database using heuristic search method. However, they take an exponential increase of runtime as a minimum share threshold decreased. Some techniques were presented to mine all share-frequent itemsets, however, these methods cannot extract the complete set of share-frequent itemsets since their models do not satisfy the downward closure property. These include SIP [15], CAC [16] and IAB [11].

Later, the Fast Share Measure approach (ShFSM) was proposed to improve the previous approaches by using a property of level closure. However, this model does not hold the property of downward closure, therefore, the set of share-frequent

itemsets cannot be discovered completely. After that, [17] proposed the Direct Candidates Generation(DCG) that relies on the downward-closure-property by using the transaction measure value of a pattern (definition 4) to decrease the number of useless candidates. This method adopts the level-wise candidate generation-and-test methodology that directly generates candidates without joining and pruning steps in each pass. Moreover, this method extracts the complete set of share-frequent pattern. However, the DCG requires more execution time to generate and test a large number of unwanted candidates in the mining process since it treats all *1*-items from the transaction database as *1*-candidate (C_1) for generating all *2*-candidates. That is, all *1*-candidates contain frequent itemsets and infrequent itemsets. For later round, the method produces the k-candidates (C_k) by joining the previous candidate (C_{k-1}) which has their transaction measure value beyond a certain threshold with a single pattern in C_1.

As can be seen in the problem mentioned above, reducing the computation time is an important issue of mining share-frequents patterns. [13], therefore, introduced an efficient method for Mining Complete Share-frequent Itemsets (MCShFI) to solve the weakness of the DCG solution. The MCShFI uses transaction measure value (definition 4) as an effective upper bound of each item to reduce the computation time by moving out a number of unwanted candidate at the first round. Furthermore, the generated candidate itemsets of MCShFI are created from the combination of only itemsets with their transaction measure value beyond a certain threshold. As a result, the MCShFI approach is more efficient than the DCG method in terms of execution time.

2.2 The Fast Update Concept

The number of transactions in a database is increasingly growing up; therefore, the evaluation of derived association rules must be reproduced. In fact, some association rules are newly generated while some other rules become obsolete [18]. Every time the new inserted transaction occurs in the database, in the traditional batch mining algorithms, it has to re-process the whole updated database. This process; therefore, consumes lots of execution time and additionally waste existing mined knowledge.

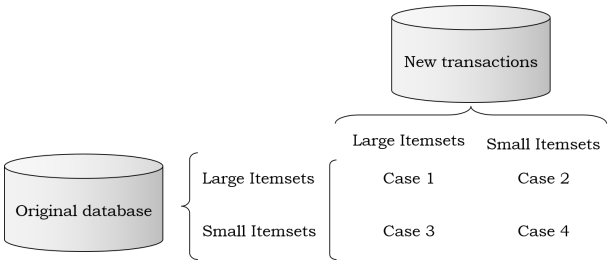


Figure 1: Four cases when new transactions are added into existing database

To reduce computation time, Cheung et al. [14] had introduced the Fast Update (FUP) algorithm. The FUP takes into account only mining results of an original database and new transactions. There are four cases under consideration as shown in Figure 1.

Case 1: An itemset is large both in an original database and in new transactions.
Case 2: An itemset is large in an original database but small in new transactions.
Case 3: An itemset is small in an original database but large in new transactions.
Case 4: An itemset is small both in an original database and in new transactions.

For Case 1, both the original database and the new transactions are large (frequent). Thus, the weighted average of the counts will be large. Identically, for Case 4, both small (infrequent) original database and small (infrequent) newly inserted transactions yield small weighted average of the counts. This implies that, both Case 1 and Case 4 will not influence the final frequent itemsets. Nevertheless, for Case 2 and Case 3, the weighted average of the counts can be small (infrequent), thus, affects the final frequent itemsets. Hence, to achieve the correct solution, the existing large (frequent) itemsets may be removed for Case 2 while it must be inserted for Case 3.

3 Problem Statement

In this section, the notation for the IS-FUP algorithm is described. Then the basic definitions which similar to the previous works [19, 17, 20, 13] are presented.

3.1 Notation

I	a set of m items, $I = \{i_1, i_2, \ldots, i_j, \ldots, i_m\}$ be a set of literals with counting attributes, where $i_1, i_2, \ldots, i_j, \ldots, i_m$ are called *item*.
X	an itemset, where $X \subseteq I$.
DB	a transaction database, $DB = \{T_1, T_2, \ldots, T_n\}$ be a set of transactions, where n is the number of transactions in DB and each transaction $T_q \in DB$ and also $T_q \subseteq I$, $(1 \leq q \leq n)$. The type of DB can be divided into two groups: O and $db+$.
O	an original transaction database.
$db+$	a set of the newly inserted transactions.
U	an entire updated database, $O \cup db+$
C_k	a candidate share-frequent itemsets of size k
$minLMV^{db+}$	The minimum local measure value in new transactions $db+$
$minLMV^{U}$	The minimum local measure value in the entire updated database U

TMV^O	The total measure value of the original transaction database O
TMV^{db+}	The total measure value of the newly inserted transactions $db+$
TMV^U	The total measure value of the entire updated database U
$Htmv^O$	a set of high transaction measure value itemsets in the original transaction database O
$Htmv^U$	a set of high transaction measure value itemsets in the entire updated database U
$Htmv_k^O$	a set of high transaction measure value k-itemsets in the original transaction database O
$Htmv_k^{db+}$	a set of high transaction measure value k-itemsets in the newly inserted transactions $db+$
$Htmv_k^U$	a set of high transaction measure value k-itemsets in the entire updated transactions U
$tmv^O(X)$	a transaction measure value of an itemset X of all transactions in the original transaction database O
$tmv^{db+}(X)$	a transaction measure value of an itemset X of all transactions in the newly inserted transactions $db+$
$tmv^U(X)$	a transaction measure value of an itemset X of all transactions in the entire updated transactions U
$lmv^O(X)$	a local measure value of all transactions in the original transaction database O that containing X
$lmv^{db+}(X)$	a local measure value of all transactions in the newly inserted transactions $db+$ that containing X
$lmv^U(X)$	a local measure value of all transactions in the entire updated transactions U that containing X

Table 1: Example of a transaction database with counting

	TID	Transaction	Count
Original Database (O)	T_1	{A,B,F,G,H}	{2,1,2,1,2}
	T_2	{A,C,E}	{5,3,3}
	T_3	{B,C,H}	{3,2,2}
	T_4	{C}	{5}
	T_5	{C,E,F,G}	{3,1,2,1}
The newly inserted transactions ($db+$)	T_6	{A,C,D,E,F}	{4,2,1,1,5}
	T_7	{A,D}	{1,2}
	T_8	{B,C,H}	{4,3,2}
	T_9	{A,C,E}	{2,1,1}

3.2 Basic Definitions

Definition 3.1. The measure value of item i_p in transaction T_q represents a numerical value associated with item i_p in T_q denoted by,

$$mv(i_p, T_q). \tag{3.1}$$

For example, in Table 1, $mv(\{C\}, T_2)$ =3.

Definition 3.2. The itemset measure value $imv(X, T_q)$ is the *total measure value* of itemset X in transaction T_q defined as follow,

$$imv(X, T_q) = \sum_{i_p \in X} mv(i_p, T_q). \tag{3.2}$$

For instance in Table 1, $imv(\{AC\}, T_2) = mv(\{A\}, T_2) + mv(\{C\}, T_2) = 5{+}3 = 8$.

Definition 3.3. A transaction measure value of transaction T_q, $tmv(T_q)$ is defined by,

$$tmv(T_q) = \sum_{i_p \in T_q} mv(i_p, T_q). \tag{3.3}$$

For example in Table 1, $tmv(T_2) = mv(\{A\}, T_2) + mv(\{C\}, T_2) + mv(\{E\}, T_2) = 5{+}3{+}3 = 11$.

Definition 3.4. A transaction measure value of itemset X denoted by $tmv(X)$ describes a total value of all transactions containing X. It is defined by,

$$tmv(X) = \sum_{X \subseteq T_q \in DB_X} tmv(T_q) \tag{3.4}$$

where DB_X is a set of transaction that contains itemset X.

For example, $tmv(\{AC\}) = tmv(T_2) + tmv(T_6) + tmv(T_9) = 11 + 13 + 4 = 28$, in Table 1.

Definition 3.5. A local measure value of itemset X is given by,

$$lmv(X) = \sum_{T_q \in DB_X} \sum_{i_p \in X} imv(i_p, T_q). \tag{3.5}$$

For example in Table 1, $lmv(\{AC\}) = imv(\{AC\}, T_2) + imv(\{AC\}, T_6) + imv(\{AC\}, T_9) = 8 + 6 + 3 = 17$.

Definition 3.6. A total measure value of the entire updated transaction, $TMV(U)$, represents the summation of all the transaction measure values which is given by,

$$TMV(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q). \tag{3.6}$$

For example in Table 1, $TMV(O \cup db+) = 67$.

Definition 3.7. A share value of itemset X, denoted by $SH(X)$, is a ratio of the local measure value of X to the total measure value of DB, as calculated by,

$$SH(X) = \frac{lmv(x)}{TMV(U)}. \tag{3.7}$$

For example, $SH(\{AC\}) = = 17/67 = 0.2537$.

Definition 3.8. Given that a minimum share ($minShare$) is a minimum share threshold, if the share value of itemset X, $SH(X)$, is greater than $minShare$ then we can conclude that itemset X is a share-frequent itemset.

For the example database, if $minShare$ is 0.25, $\{AC\}$ is a share-frequent itemset, as $SH(\{AC\}) = 0.2537$.

Definition 3.9. A minimum local measure value ($minLMV$) can be defined by,

$$minLMV = ceiling(minShare \times TMV(O \cup db+)) \tag{3.8}$$

Therefore, in Table 1, $minLMV = ceiling(0.25 \times 67) = 17$. For any itemset X, if $lmv(X) \geq minLMV$, then itemset X is a share-frequent itemset.

From the property of the downward closure (also called anti-monotonicity) of the support measure, it can be seen that this property can maintain in the share-frequent patterns mining by using the tmv value of itemset X (Definition 4). For any itemset X, if $tmv(X)$ is less than $minLMV$, all supersets of X can be pruned (including X) immediately without further consideration. For example, let minimum share threshold is 0.25, then $tmv(\{D\}) = 16 < minLMV = 17$. According to the property of downward closure, all supersets of $\{D\}$ are not generated as a candidate patterns. Therefore, we can prune $\{D\}$ at the early step.

4 A Proposed Incremental Algorithm

We propose an efficient algorithm for incremental share-frequent itemsets mining based on a Patternset Table Knowledge (PSTable knowledge). The proposed algorithm includes two processes: maintenance a PSTable knowledge, and the IS-FUP algorithm.

4.1 Maintenance a PSTable Knowledge

A PSTable knowledge is a set of itemsets with their count information and it is im-proved by the concept of a bittable structure [21, 22, 23, 20, 13]. This structure can avoid candidate generation by aggregate the transactions that have similar itemsets. In addition, the PSTable knowledge will be constructed once and it accommodates modified data in the future (do not need to rebuild). The following describes operation of this algorithm.

For creating a PSTable knowledge and inserting new transactions, the algorithm reads the data from each incoming transaction. After that, it checks the itemsets of the transaction whether the itemset is already available in the PSTable or not. If it exists, the quantity of the transaction (total count) is summed with the new entries to gain new value of total count, and the itemset count is incremented by 1. If the itemset is new, it will be inserted into the PSTable with the quantity of the transaction and set the itemset count to be 1. This process is repeated for all transactions.

For maintenance the PSTable knowledge (removing transactions), the algorithm reads the itemset from each removing transaction. After that, the itemset is searched from the PSTable knowledge, then the quantity of each item of the itemset and the quantity of the transaction (total count) are updated and also decrease the itemset count by 1. For the modified existing transactions case, the algorithm starts with ascendant sorting items in the PSTable knowledge. Next, the algorithm discovers the modified transaction and then the quantity of each item of the itemset and the quantity of the transaction (total count) are updated. All the mentioned steps are repeated for all of the transactions in database. For instance, the original transaction database in Table 1 was performed and the results shown in Table 2.

Table 2: The PSTable after the algorithm executed all transactions

PID	A	B	C	D	E	F	G	H	Total Count	Itemset Count
1	2	1	0	0	0	2	1	2	8	1
2	7	0	4	0	4	0	0	0	15	2
3	0	7	5	0	0	0	0	4	16	2
4	0	0	5	0	0	0	0	0	5	1
5	0	0	3	0	1	2	1	0	7	1
6	4	0	2	1	1	5	0	0	13	1
7	1	0	0	2	0	0	0	0	3	1

4.2 Incremental Share-Frequent Itemset Mining

4.2.1 The Concept of the IS-FUP Algorithm

We propose an efficient algorithm that handles insertion of transactions (named IS-FUP algorithm). It is extended from the MCShFI algorithm [13] in a batch way for discovering complete share-frequent itemsets from a PSTable knowledge based on the FUP concept [14]. Assuming that a PSTable knowledge is built in advance from the original database before the new transactions come and the large/frequent itemsets have been already derived from the initial database. In IS-FUP algorithm, itemsets are separated into four scenarios according to whether they are large (frequent) or small (infrequent) itemsets for both in the original database and in the new transactions. To determine whether the itemset is large or small, the value of tmv is calculated (by definition 3.4). As mentioned earlier, this value contains the properties of downward closure. If the value of tmv of any itemset is larger than or equal to the minimum local measure value ($minLMV$), it is in the set of high transaction measure value itemsets and in the group of large itemsets. Otherwise, it is in small itemsets.

Considering an original database and a new transaction that will be inserted, the following four cases and their results in given in Table 3.

Table 3: Example of a transaction database with counting

Cases : Original-New transactions	Results
Case 1: Large itemset - Large itemset	Always large itemset
Case 2: Large itemset - Small itemset	Determined by rescanning the newly added transactions
Case 3: Small itemset - Large itemset	Determined by rescanning the PSTable
Case 4: Small itemset - Small itemset	Always small itemset

For the newly inserted transactions, when some new records are inserted into the original database: In Case 1 and 4, itemsets do not change the final share-frequent itemsets and thus these two cases are extreme. In Case 2, itemsets may be removed from existing large itemsets, and itemsets in Case 3 may add new large itemsets. For the extreme cases 1 and 4, the proposed approach in share-frequent itemset mining can be mathematically proved and given below whereas the other two cases 2 and 3 can be shown by experiments using the IS-FUP algorithm. Also, a numerical example is later provided in the next sections illustrating all the four cases.

Theorem 4.1. *If an itemset X is large in both the original database and the newly inserted transactions then the itemset X is large in the entire updated database (Case 1).*

Proof. Assume, by contradiction, that itemset X is not large in the entire updated

database, i.e.,

$$tmv(X)^O + tmv(X)^{db+} < minShare \times (TMV^O + TMV^{db+}) \tag{4.1}$$

and so

$$tmv(X)^O + tmv(X)^{db+} - minShare \times TMV^{db+} < minShare \times TMV^O \tag{4.2}$$

By hypothesis, an itemset X is large in both the original database and the newly inserted transactions

it is obtained that

$$tmv(X)^O \geq minLMV^O (= minShare \times TMV^O) \tag{4.3}$$

and

$$tmv(X)^{db+} \geq minLMV^{db+} (= minShare \times TMV^{db+}). \tag{4.4}$$

Since

$$tmv(X)^O < tmv(X)^O + tmv(X)^{db+} - minShare \times TMV^{db+}. \tag{4.5}$$

From (4.2) and (4.5), we can conclude that

$$tmv(X)^O < minShare \times TMV^O \tag{4.6}$$

(4.6) contradicts (4.3), hence the proof is complete. □

Theorem 4.2. *If an itemset X is small in both the original database and the newly inserted transactions then the itemset X is small in the entire updated database (Case 4).*

Proof. Assume, by contradiction, that itemset X is not small in the entire updated database, i.e.,

$$tmv(X)^O + tmv(X)^{db+} \geq minShare \times (TMV^O + TMV^{db+}) \tag{4.7}$$

and so

$$tmv(X)^O + tmv(X)^{db+} - minShare \times TMV^{db+} \geq minShare \times TMV^O. \tag{4.8}$$

By hypothesis, an itemset X is small in both the original database and the newly inserted transactions

it is obtained that

$$tmv(X)^O < minLMV^O (= minShare \times TMV^O) \tag{4.9}$$

and

$$tmv(X)^{db+} < minLMV^{db+} (= minShare \times TMV^{db+}). \tag{4.10}$$

Since

$$tmv(X)^O > tmv(X)^O + tmv(X)^{db+} - minshare \times TMV^{db+}. \quad (4.11)$$

From (4.8) and (4.11), we can conclude that

$$tmv(X)^O \geq minShare \times TMV^O \quad (4.12)$$

(4.9) contradicts (4.12), hence the proof is complete. □

4.2.2 The IS-FUP Algorithm

The detail of the proposed algorithm are described step by step below.

Input: 1) A PSTable knowledge (O) 2) The newly inserted transactions ($db+$) in which each transaction includes items and their quantities 3) A $Htmv^O$, which is stored a set of high transaction measure value of the original transaction database 4) The total measure value of the original transaction database, TMV^O 5) The minimum share threshold, $minShare$

Output: A $Htmv^U$, that is a set of high transaction measure value of the entire updated database.

Step 1: Calculate the total measure value of newly inserted transactions using Definition 3.6 denoted by TMV^{db+}. After that, TMV^U is calculated ($TMV^U = TMV^O + TMV^{db+}$). Next, the algorithm computes $minLMV^U$ using Definition 3.9 and set $k = 1$, where k is used for recording the number of items in the itemsets currently being processed.

Step 2: The algorithm generates the candidated k-itemsets then computes their transaction measure value $tmv^{db+}(X)$ and local measure value $lmv^{db+}(X)$ from the new transactions using Definition 3.4 and 3.5 respectively. Checks whether the $tmv^{db+}(X)$ of each k-itemset X is larger than or equal to $minLMV^{db+}$. If $tmv^{db+}(X)$ satisfies the above condition, put X in the set of high transaction measure value k-itemset in the newly inserted transactions, $Htmv_k^{db+}$.

Step 3: For each k-itemset X, if it is in the set of high transaction measure value in the new transactions and it also appears in the set of high transaction measure value in the original database , do the following substeps (Case 1):

Step 3.1: The algorithm will keep the itemset X into both the $Htmv_k^U$ and C_k and also removes it from the $Htmv_k^O$.

Step 4: For each k-itemset X, if it is in the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$) but it does not appear in the set of high transaction measure value in the original database $Htmv_k^O$, do the following substeps (Case 3):

Step 4.1: Rescans the original database to determine the $tmv^O(X)$ and $lmv^O(X)$ of k-itemset X. After that, set both $tmv^U(X)$ $(= tmv^O(X) + tmv^{db+}(X))$ and $lmv^U(X)$ $(= lmv^O(X) + lmv^{db+}(X))$.

Step 4.2: Checks whether the $tmv^U(X)$ is greater than or equal to $minLMV^U$. If $tmv^U(X)$ satisfies the above condition, stores it to the C_k and also adds into the $(Htmv_k^U)$.

Step 5: For each k-itemset X, if it appears in the set of high transaction measure value in the original database $(Htmv_k^O)$ but does not appear in the set of high share-frequent itemsets from the new transactions $(Htmv_k^{db+})$, do the following substeps (Case 2):

Step 5.1: The algorithm reads all k-itemsets in the $Htmv_k^O$. After that, compute the tmv^{db+} and the lmv^{db+} from the PSTable knowledge and then calculates both $tmv^U(X)$ $(= tmv^O(X) + tmv^{db+}(X))$ and $lmv^U(X)$ $(= lmv^O(X) + lmv^{db+}(X))$.

Step 5.2: If the $tmv^U(X)$ is larger than or equal to $minLMV^U$, stores X to the C_k and inserts to the $Htmv_k^U$ and also removes it from the $Htmv_k^O$.

Step 6: Increase k by one and then the algorithm generates the candidate k-itemsets from the variable C_{k-1}.

Step 7: Repeat Steps 2 - 6 until no new candidate itemsets is generated and the $Htmv^O$ is empty.

4.2.3 A Numerical Example

This subsection shows how the IS-FUP algorithm can be used for finding out the share-frequent itemsets from the new transactions. Assume that the transactions $T_1 - T_5$ shown in Table 1 are the initial database and the transaction T_6 - T_9 are newly inserted transactions. Also assume that the minimum share threshold $(minShare)$ is set at 0.25 which is 25% of total quantity. The set of high transaction measure value is built in advance from the original database as shown in Table 4.

Step 1: : The IS-FUP algorithm calculates the initial variables at first i.e. TMV^{db+}, $minLMV^{db+}$, TMV^U and $minLMV^U$ respectively, as shown in Table 5. Then, k is set at 1.

Step 2: The k-itemsets in the newly inserted transactions for $\{A\}, \{B\}, \{C\}, \{D\}$, $\{E\}, \{F\} and \{H\}$ are read into the algorithm and then calculate their transaction measure value (tmv^{db+}) and the local measure value (lmv^{db+}) from the new transactions. Take k-itemset $\{A\}$ as an example to illustrate the process. The k-itemset $\{A\}$ appears in the new transactions

Table 4: The $Htmv^O$

Itemset X	$tmv^O(X)$	$lmv^O(X)$
$\{A\}$	19	7
$\{B\}$	15	4
$\{C\}$	30	13
$\{E\}$	18	4
$\{F\}$	15	4
$\{G\}$	15	2
$\{H\}$	15	4
$\{A, C\}$	11	8
$\{A, E\}$	11	8
$\{B, H\}$	15	8
$\{C, E\}$	18	10
$\{F, G\}$	15	6
$\{A, C, E\}$	11	11

T_6, T_7 and T_9 and then the algorithm calculates the $tmv^{db+}(\{A\})$ as (13+3+4), which is 20 and computes the $lmv^{db+}(\{A\})$ as (4+1+2), which is 7 respectively. The other k-itemsets are calculated in the same way and the result is shown in Table 6. After that, checks whether the $tmv^{db+}(X)$ of each k-itemset is larger than or equal to $minLMV^{db+}$. In this example, all k-itemsets satisfy the condition and then put them in the set of high transaction measure value k-itemsets in the new transaction, ($Htmv_k^{db+}$).

Table 5: The initial variables for the algorithm

	TMV	$minLMV$
O	38	10
$db+$	29	7
U	67	17

Step 3: For each k-itemset X, it is in both the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$) and the set of high transaction measure value in the original database ($Htmv_k^O$), do the following substeps (Case 1):

Step 3.1: The k-itemsets are in the $Htmv_k^O$ and the $Htmv_k^{db+}$ which are as follows $\{A\}, \{B\}, \{C\}, \{E\}, \{F\} and \{H\}$. These itemsets are recorded to the $Htmv_k^U$ and are removed from the $Htmv_k^O$ and also kept as candidate share-frequent itemsets, C_k. The result is shown in Table 7 and 8.

Table 6: all k-itemsets are calculated from the new transactions

k-itemset X	$tmv^{db+}(X)$	$lmv^{db+}(X)$
$\{A\}$	20	7
$\{B\}$	9	4
$\{C\}$	26	6
$\{D\}$	16	3
$\{E\}$	17	2
$\{F\}$	13	5
$\{H\}$	9	2

Table 7: After k-itemsets (Case 1) are added in the $Htmv^U$

k-itemset X	$tmv^U(X)$	$lmv^U(X)$
$\{A\}$	39	14
$\{B\}$	24	8
$\{C\}$	56	19
$\{E\}$	35	6
$\{F\}$	28	9
$\{H\}$	24	6

Table 8: After k-itemsets (Case 1) are deleted from the $Htmv^O$

k-itemset X	$tmv^O(X)$	$lmv^O(X)$
$\{G\}$	15	2
$\{A, C\}$	11	8
$\{A, E\}$	11	8
$\{B, H\}$	15	8
$\{C, E\}$	18	10
$\{F, G\}$	15	6
$\{A, C, E\}$	11	11

Step 4: For each k-itemset X, it is in the set of high transaction measure value in the newly inserted transactions, $Htmv_k^{db+}$, but it does not in the set of high transaction measure value in the original database, $Htmv_k^O$, do the following substeps (Case 3):

Step 4.1: The k-itemset X appears in the $Htmv_k^{db+}$ but it does not appear in the $Htmv_k^O$ which is $\{D\}$. The IS-FUP algorithm rescans to calculate the $tmv^O(X)$ and the $lmv^O(X)$ from

the PSTable knowledge. Then, the algorithm computes the $tmv^U(X)$ and the $lmv^U(X)$ respectively. Let takes the k-itemset $\{D\}$ as an example to illustrate the process. The tmv^{db+} of $\{D\}$ and the lmv^{db+} of $\{D\}$ in the new transactions are 16 and 3 respectively as shown in Table 6. The tmv^O of $\{D\}$ and the lmv^O of $\{D\}$ in the original database are both 0, which are shown in Table 9. The updated transaction measure value and The updated local measure value for the updated database of $\{D\}$ are calculated as $tmv^U(\{D\})$ (=0+16), which is 16 and $lmv^U(\{D\})$ (=0+3), which is 3 respectively.

Step 4.2: Check whether the tmv^U of k-itemset X is larger than or equal to the $minLMV^U$. If it satisfies the condition, inserts X into C_k and also stores it to the $Htmv_k^U$. For instance, the k-itemset $\{D\}$ does not satisfy the both of above conditions.

Step 5: For each k-itemset X, it does not in the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$), but it appears in the set of high transaction measure value in the original database ($Htmv_k^O$), do the following substeps (Case 2):

Step 5.1: The IS-FUP algorithm reads all k-itemsets from the $Htmv_k^O$ in Table 8, which is $\{G\}$. The tmv^{db+} and the lmv^{db+} of k-itemset $\{G\}$ in the new transactions are calculated, which are both 0. Then, the updated transaction measure value and the updated local measure value for the updated database of $\{G\}$ are computed as $tmv^U(\{G\})$ (=0+15), which is 15 and $lmv^U(\{G\})$ (=0+2), which is 2 respectively. The result is shown in Table 10.

Step 5.2: Check whether the tmv^U of $\{G\}$ is larger than or equal to the $minLMV^U$. If it satisfies the condition, stores it into both C_k and the $Htmv_k^U$ and also removes it from the $Htmv_k^O$. For instance, the k-itemset $\{G\}$ does not satisfy both of the above condition.

Table 9: After all k-itemsets in case 3 are re-calculated.

k-Itemset X	$tmv^O(X)$	$lmv^O(X)$	$tmv^U(X)$	$lmv^U(X)$
$\{D\}$	0	0	16	3

Step 6: Set $k = 2$ $(k + 1)$. The candidate k-itemsets are generated from C_{k-1}, which are $\{A, B\}$, $\{A, C\}$, $\{A, E\}$, $\{A, F\}$, $\{A, H\}$, $\{B, C\}$, $\{B, E\}$, $\{B, F\}$, $\{B, H\}$, $\{C, E\}$, $\{C, F\}$, $\{C, H\}$, $\{E, F\}$, $\{E, H\}$, $\{F, G\}$ and $\{F, H\}$.

Table 10: After all k-itemsets in case 2 are re-calculated.

k-Itemset X	$tmv^{db+}(X)$	$lmv^{db+}(X)$	$tmv^{U}(X)$	$lmv^{U}(X)$
$\{G\}$	0	0	15	2

Step 7: Steps 2 - 6 are then repeated until no candidate itemset is generated and the $Htmv^{O}$ is empty as shown in Table 12. The IS-FUP algorithm returns a set of high transaction measure value in the entire updated database, $Htmv^{U}$, that contains 3 share-frequent itemsets i.e., $\{C\}$, $\{A, C\}$ and $\{A, C, E\}$ that shown in Table 11.

Table 11: The $Htmv^{U}$ after the IS-FUP completed

Itemset X	$tmv^{U}(X)$	$lmv^{U}(X)$	$SH(X)$
$\{A\}$	39	14	0.2090
$\{B\}$	24	8	0.1194
{C}	**56**	**19**	**0.2836**
$\{E\}$	35	6	0.0896
$\{F\}$	28	9	0.1343
$\{H\}$	24	6	0.0896
{A,C}	**28**	**17**	**0.2537**
$\{A, E\}$	28	16	0.2388
$\{A, F\}$	21	13	0.1940
$\{B, H\}$	24	14	0.2090
$\{C, E\}$	35	15	0.2239
$\{C, F\}$	20	12	0.1791
$\{E, F\}$	20	9	0.1343
{A,C,E}	**28**	**22**	**0.3284**
$\{C, E, F\}$	20	14	0.2090

Table 12: The $Htmv^{O}$ after the proposed algorithm completed

Itemset X	$tmv^{O}(X)$	$lmv^{O}(X)$
– Empty –		

5 Algorithm Efficiency

In this section, we demonstrate the efficiency of the algorithm by applying the IS-FUP to two IBM synthetic datasets (T10I4D100K and T40I10D100K) and a real-life dataset (pumsb*). They are acquired from frequent itemset mining dataset repository page (http://fimi.ua.ac.be/data/). As like the performance evaluation of the previous share-frequent itemset mining algorithm [17, 13], we modified these datasets by creating random numbers for the quantity of each item in each transaction, ranging from 1 to 10. The characteristics of the datasets are shown in Table 13, $|D|$ is the total number of transaction in a dataset, $|N|$ is the number of distinct pattern in the dataset and T_{avg} is the average size of transaction. Our algorithms were written in Microsoft C++ 2010 and carried out on a PC with 1.40 GHz CPU with 4 GB main memory. The algorithm efficiency is determined by the processing time and compared to MCShFI [13] in the batch way.

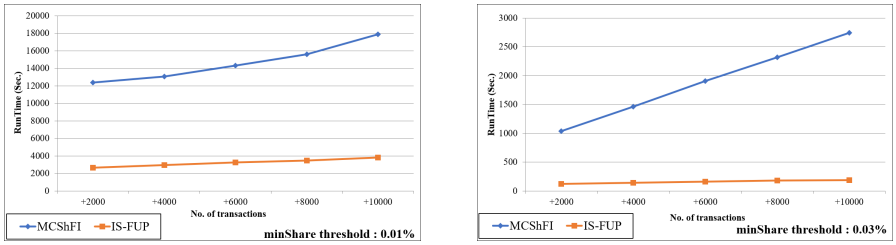
Table 13: Characteristics of the experiment datasets

Dataset	Size (MB)	$\|D\|$	$\|N\|$	T_{avg}
T10I4D100K	3.83	100,000	870	10.1
T40I10D100K	14.7	100,000	942	40.5
pumsb*	10.7	49,046	2,088	50.5

5.1 The IS-FUP Algorithm on Synthetic Datasets

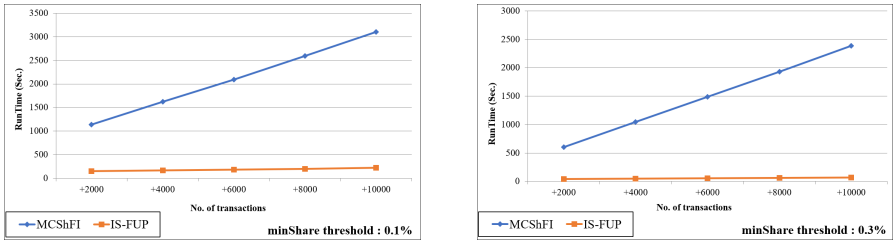
In the first experiment, we have tested the effectiveness of the IS-FUP algorithm in incremental mining and the batch MCShFI with T10I4D100K. The minimum share threshold was set at 0.01% and 0.03% respectively. A number of 70,000 transactions were created to initially mine the share-frequent itemsets with their quantities. Next, the numbers of inserted transactions were varied from 2,000 to 10,000 and the results are shown in Figures 2 (a)-(b). For T40I10D100K, experiments were conducted in similar to the previous experiment but the minimum share threshold was set at 0.1% and 0.3% respectively. The experimental results are illustrated in Figures 3 (a)-(b).

It can be easily observed from these figures that the both algorithms required more execution time when the certain threshold lowers. This is reasonable because when the minimum share value becomes minor, the algorithms have to generate more candidates in the mining process. Moreover, the runtime of the new solution is smaller than the MCShFI method in a batch way. This is because the MCShFI needs to update information on its data structure to meet updated transactions. After that, it has to mine the last updated database to get the final share-frequent itemsets. Therefore, this process must be performed every time whenever transactions are inserted. For the new solution, it takes advantages of the earlier result of share-frequent itemsets and does not re-mine the whole updated database. The



(a) (b)

Figure 2: Effects of numbers of inserted transactions on runtime for the T10I4D100K dataset



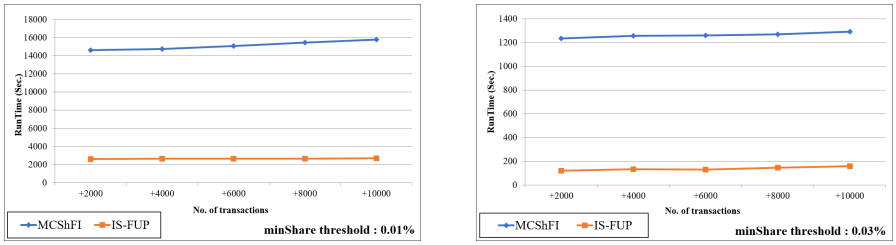
(a) (b)

Figure 3: Effects of numbers of inserted transactions on runtime for the T40I10D100K dataset

proposed method mines only the newly inserted transactions and rescans as necessary to re-calculate itemsets from the original database only if it is in Case 3 as mentioned in Section 4.2.1.

In the next experiments, the first 90,000 transactions were discovered from the T10I4D100K dataset to construct initial share-frequent itemsets with their quantities. The minimum share threshold was set at 0.01% and 0.03% respectively. After that, each 2,000 transactions, which were used as new inserted transactions, were performed again with the same thresholds. The results are shown in Figures 4 (a)-(b). Similar to above experiments for T40I10D100K, the minimum share threshold was set at 0.1% and 0.3% respectively. The experimental results are illustrated in Figures 5 (a)-(b).

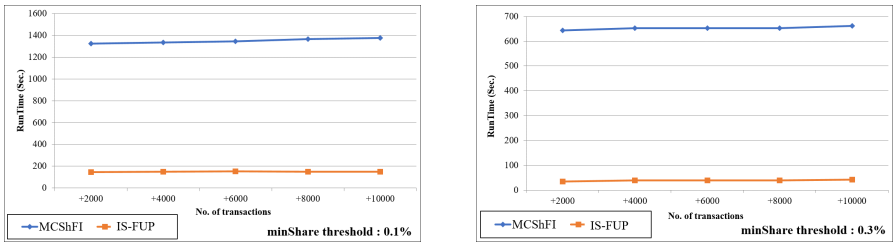
Figure 6 (a) is then made to show the runtimes of the two algorithms on the T10I4D100K dataset for different minimum share threshold. The thresholds range of 0.005 to 0.05 percent were used here. The first 90,000 transactions were mined from the T10I4D100K dataset to create initial share-frequent itemsets with their quantities. After that, the next 10,000 transactions, which were used as new inserted transactions, were generated with the same thresholds. In the similar way



(a) (b)

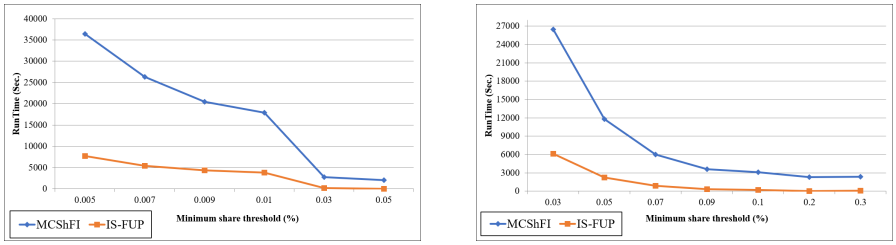
Figure 4: Effects of numbers of inserted transactions on runtime for the T10I4D100K dataset

for T40I10D100K, the minimum share thresholds were set from 0.03% to 0.3%. The experimental results illustrated in Figure 6 (b).



(a) (b)

Figure 5: Effects of numbers of inserted transactions on runtime for the T40I10D100K dataset

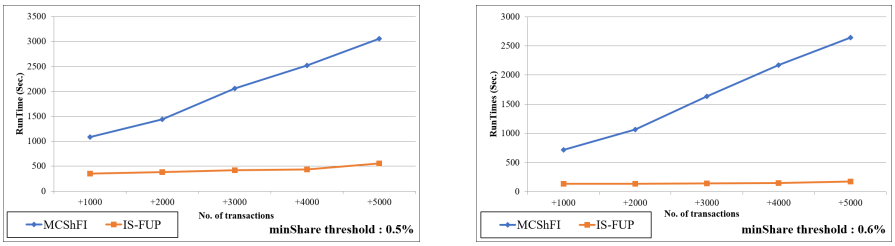


(a) (b)

Figure 6: Effects of different minimum share thresholds on runtime for the T10I4D100K and the T40I10D100K datasets

5.2 The IS-FUP Algorithm on Real-Life Dataset

At first, we constructed to initially mine the share-frequent itemsets with their quantities from the pumsb* dataset for 34,046 transactions with three different minimum share thresholds. The thresholds were set at 0.5% and 0.6% respectively. In addition, various number of transactions of 1,000, 2,000, 3,000, 4,000 and 5,000 were added and the results are shown in Figures 7 (a)-(b). As shown in these figures, the batch MCShFI algorithm wasted a lot of time to mine all of the whole updated transactions whenever transactions were inserted. The new solution does not re-mine the last updated database and hence takes advantages of the previous results of share-frequent itemsets.

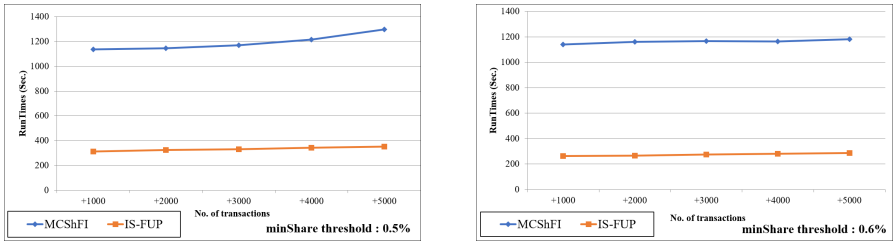


(a) (b)

Figure 7: Effects of numbers of inserted transactions on runtime for the pumsb* dataset

In the next experiments, the first 44,046 transactions were searched from the pumsb* to create initial share-frequent itemsets with their quantities. The minimum share threshold was set at 0.5% and 0.6% respectively. After that, each 1,000 transactions, which were used as new inserted transactions were generated again with the same threshold. The results are shown in Figures 8 (a)-(b).

The last experiment was then conducted to compare the runtimes of both algorithms on the pumsb* real-life dataset for different numbers of minimum share threshold. Additionally, the minimum share threshold was considered to be varied from 0.45% to 0.6%. A number of 44,046 transactions were extracted to construct initial share-frequent itemsets with their quantities. After that, the next 5,000 transactions were added and, with the same threshold value, the runtime was measured. The result is as shown in Figure 9. The computation time of the new solution is lower than the MCShFI in batch mode for handling transaction insertion. The reason is as same as already mentioned in the previous experiment. In conclusion, our proposed algorithm in incremental share-frequent itemset mining (the IS-FUP algorithm) outperforms the batch MCShFI method to maintain the updated database.



(a) (b)

Figure 8: Effects of numbers of inserted transactions on runtime for the pumsb* dataset

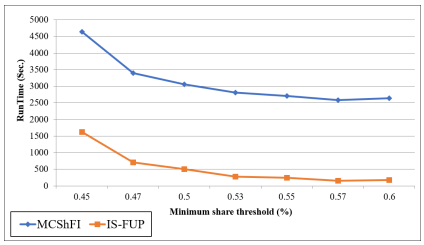


Figure 9: Effects of different minimum share thresholds on runtime for the pumsb* dataset

6 Conclusions

In the past, the efficient algorithm for mining complete share-frequent itemsets (called MCShFI) was proposed to discover complete share-frequent itemsets based on a batch approach. In this paper, an incremental approach to the problem is proposed with the fast update concept to improve mining execution time. Among the four updated cases under consideration of the fast update concept, we mathematically show that two of them do not affect mining result. This even more helps in improving execution time of the proposed algorithm. Extensive performance analyses show that execution time of the incremental algorithm outperforms of the batch one on both synthetic datasets and real-life datasets.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rule between sets of items in large database, In Proceedings of ACM SIGMOD on Management

of Data (1993), 207-216.

[2] R. Agarwal, R. Srikant, Fast algorithms for mining association rules in large databases, In Proceedings of 20th International Conference on Very Large Data Bases (1994), 487-499.

[3] J.S. Park, M.S. Chen, P.S. Yu, Using a Hash-Based method with transaction trimming for mining association rules, IEEE Transactions on Knowledge and Data Engineering 9 (1997), 813-825.

[4] S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, In Proceedings of the ACM SIGMOD on Management of Data (1997), 255-264.

[5] E. Houda, E.F. Mohamed, E.M. Mohammed, A novel approach for mining frequent itemsets: AprioriMin, In Proceedings of 4th IEEE International Colloquium on Information Science and Technology (2016), 286-289.

[6] D.P. Shubhangi, R.D. Ratnadeep, K.D. Kirange, Adaptive Apriori Algorithm for frequent itemset mining, In Proceedings of International Conference System Modeling & Advancement in Research Trends (2016), 7-13.

[7] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, In Proceedings of the ACM SIGMOD on Management of Data (2000), 1-12.

[8] J. Pei, J. Han, H. Lu, Hmine: Hyper-structure mining of frequent patterns in large database, In Proceedings of International Conference on Data Mining (2001), 441-448.

[9] R. Agarwal, C. Aggarwal, V.V.V. Prasad, A tree projection algorithm for generation of frequent itemsets, Journal of Parallel and Distributed 61 (2001) 350-371.

[10] M. El-Hajj, O.R. Zaiane, COFI approach for mining frequent itemsets revisited, In Proceeding of the ACM SIGMOD on Data Mining and Knowledge Discovery (2004), 70-75.

[11] B. Barber, H.J. Hamilton, Extracting share frequent itemset with infrequent subsets, Data mining and Knowledge Discovery 7 (2003) 153-185.

[12] C.L. Carter, H.J. Hamilton, N. Cercone, Share based measures for itemsets, In Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery (1997), 14-24.

[13] C. Nawapornanan, V. Boonjing, An efficient algorithm for mining complete share-frequent itemsets using BitTable and heuristics, In Proceedings of International Conference on Machine Learning and Cybernetics (2012), 96-101.

[14] D.W. Cheung, J. Han, V. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: An incremental updating technique, In Proceedings of International Conference on Data Engineering (1996), 106-114.

[15] B. Barber, H.J. Hamilton, Algorithms for mining share frequent itemsets containing infrequent subsets, In Proceedings of Fourth European Conference on Principles of Knowledge Discovery in Database (2000), 316-324.

[16] B. Barber, H.J. Hamilton, Parametric algorithm for mining share frequent itemsets, Journal of Intelligent Information Systems 16 (2001) 277-293.

[17] Y.-C. Li, J.-S. Yeh, C.-C. Chang, Direct candidates generation: a novel algorithm for discovering complete share-frequent itemsets, In Proceedings of International Conference of Fuzzy Systems and Knowledge Discovery (2005), 551-560.

[18] T.P. Hong, C.W. Lin, Y.L. Wu, Incrementally fast updated frequent pattern trees, Expert Systems with Applications 34 (2008) 2424-2435.

[19] Y.-C. Li, J.-S. Yeh, C.-C. Chang, A fast algorithm for mining share-frequent itemsets, In Proceedings of International Conference on Asia-Pacific Web (2005), 417-428.

[20] C. Nawapornanan, V. Boonjing, A new share frequent itemsets mining using incremental BitTable knowledge, In Proceedings of 5th International Conference on Computer Sciences and Convergence Information Technology (2011), 358-362.

[21] J. Dong, M. Han, An efficient mining frequent itemsets algorithm, Knowledge-Based Systems. 20 (2007) 329-335.

[22] H. Jin, A counting mining algorithm of maximum frequent itemset based on matrix, In Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery (2010), 1418-1422.

[23] N. Khare, N. Adlakha, K.R. Paradasani, An algorithm for mining multidimensional association rules using boolean matrix, in Proceedings of International Conference on Recent Trends In Information, Telecommunication and Computing (2010), 95-99.

(Received 28 September 2017)
(Accepted 26 April 2018)

Appendix B

Conference Publication

HCG: A New Algorithm for Mining Share-Frequent Patterns

Chayanan Nawapornanan[1], Veera Boonjing[2]
[1]Department of Computer Science, Faculty of Science,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok 10520, Thailand
[2]International College,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok 10520, Thailand
Email: s2650854@kmitl.ac.th, kbveera@kmitl.ac.th

***Abstract*— The paper proposes a new efficient algorithm, named HCG algorithm, to mine share-frequent patterns from an incremental pattern set table knowledge called PSTable. The PSTable stores all non redundant patterns with their count information by a single database scan. A transaction newly added to the database can be incrementally added to the PSTable. The new algorithm efficiently discovers all share-frequent patterns from the PSTable by generating candidates from only high share atomic patterns. Its correctness is assured by the downward closure property. The experiment results on dense and sparse datasets show that the proposed algorithm is more efficient than existing algorithms in terms of both execution time and number of candidates.**

Keywords—Data mining; share-frequent patterns mining; frequent pattern mining.

I. INTRODUCTION

Mining frequent pattern is a crucial process and occurs in many real-world applications. These applications include association rule analysis, sequential patterns mining, correlations analysis, multi-dimensional patterns mining and etc. It was firstly introduced in 1993 by [1]. Many algorithms for frequent pattern mining have been proposed. They can be divided into two main groups: Apriori-based and pattern growth-based algorithms. Apriori-based algorithms [2-4] use a generate-and-test approach to the problem. Pattern growth-based algorithms adopt a divide-and-conquer method with an FP-Tree structure [5-8].

The above algorithms are based on support-confidence framework that treats an item in a database as a binary value. In real-world behavior, customers can buy an item in more than one piece. Hence, the traditional framework can not reflect exact number of purchased items. In order to overcome the limitation of frequent pattern mining, a share-confidence framework was defined by [9]. This model investigates useful information about numerical values correlated with items in a transaction database. The knowledge can reflect the profit or cost of the items [10]. There are many approaches to problem of share- frequent pattern mining. The Zero pruning (ZP) [11] and Zero subset pruning (ZSP) [10] algorithms were proposed in 2000 and 2003 respectively. They can generate all share-frequent patterns but spend too much time on exhaustive search method. Li et al. proposed the FSM [12], the ShFSM [13] and the DCG [14] algorithms to overcome the problem. However, the FSM and the ShFSM algorithms are time-consuming on joining and pruning steps of generated candidates in each round and also generate a lot of useless candidates. The DCG algorithm improves these two algorithms by maintaining the downward closure property. It generates candidate patterns without joining and pruning steps. In 2012, [20] proposes an efficient algorithm for Mining Complete Share-frequent Itemsets (called MCShFI). It improves the DCG algorithm by pruning useless candidate patterns in early step of the algorithm. An algorithm that adopts this idea could improve its performance even more if it mines from non-redundant aggregate patterns extracted from a database. This paper proposes to use incremental PSTable to store such aggregate patterns. In addition, we present the HCG algorithm for mining share-frequent patterns from the PSTable. It limits candidates to only ones generated from high share atomic patterns.

This rest of the paper is organized as follows. Section 2 gives the problem statement and basic definitions. Section 3 describes the PSTable and the HCG algorithm - an efficient algorithm for mining share-frequent patterns from the PSTable. In section 4, the results of experiments are discussed. Finally, we conclude our study in Section 5.

II. PROBLEM STATEMENT AND DEFINITIONS

Let $I = \{i_1, i_2,\ldots,i_m\}$ be a set of literals with counting attributes, where $i_1, i_2,\ldots,i_m$ are called pattern. Let X be a patternset, where $X \subseteq I$. Let the transaction database $(DB) = \{T_1, T_2,..,T_n\}$ be a set of transaction, where n is the number of transaction in DB and each transaction $T_q \in DB$ and also $T_q \in I$, $(1 \le q \le n)$. A transaction identifier (TID) is given to each transaction. Let *min_share* be the minimum share threshold specified by users. For a patternset X, if the share value of X denoted as $X.share$ is greater than or equal to *min_share*, patternset X calls a share-frequent patternset. The definitions of share measure are presented in [13-20]. We have adopted and described with an example of a transaction database shown in Table 1.

978-1-4799-4963-2/14/$31.00 ©2014 IEEE

TABLE I. A TRANSACTION DATABASE WITH COUNTING.

TID	*Transaction*
T_1	{A:1,B:1,C:1,D:1,G:1,H:1}
T_2	{F:4,H:3}
T_3	{B:4,C:3,D:3}
T_4	{C:4,E:1}
T_5	{B:3,D:2}
T_6	{B:3,C:2,D:1}
T_7	{B:3,C:4,D:1,E:2}
T_8	{A:4,F:1,G:1}
T_9	{C:2,E:1}

Definition 1 : The measure value of pattern i_p in transaction T_q is the attribute value denoted by $mv(i_p,T_q)$. For example, $mv(\{B\},T_3) = 4$.

Definition 2 : The itemset measure value of a patternset X in transaction T_q is the total measure value of all patterns of X in T_q.

$$imv(x,T_q) = \sum_{i_p \in X} mv(i_p,T_q) \quad (1)$$

For example, $imv(\{BC\},T_3) = 7$.

Definition 3 : The transaction measure value of transaction T_q is the total measure value of a transaction T_q.

$$tmv(T_q) = \sum_{i_p \in X} mv(i_p,T_q) \quad (2)$$

For example, $tmv(T_3) = 10$.

Definition 4 : The transaction measure value of a patternset X of all transaction database containing X is denoted as,

$$tmv(x) = \sum_{x \subseteq T_q \in DB_X} tmv(T_q) \quad (3)$$

For example, $tmv(\{BC\}) = tmv(T_1) + tmv(T_3) + tmv(T_6) + tmv(T_7) = 6 + 10 + 6 + 10 = 32$.

Definition 5 : The local measure value of a patternset X in a transaction database containing X is denoted by,

$$lmv(x) = \sum_{T_q \in DB_x} \sum_{i_p \in X} imv(i_p,T_q) \quad (4)$$

For example, $lmv(\{BC\}) = imv(\{BC\},T_1) + imv(\{BC\},T_3) + imv(\{BC\},T_6) + imv(\{BC\},T_7) = 2 + 7 + 5 + 7 = 21$.

Definition 6 : The total measure value of database is the total measure value of a transaction database DB.

$$TMV(DB) = \sum_{T_q \in U} \sum_{i_p \in T_q} mv(i_p,T_q) \quad (5)$$

For example, $TMV(DB) = 58$.

Definition 7 : The patternset share value of a patternset X is the ratio of the local measure value of X to the total measure value of a transaction database DB. That is,

$$SH(X) = \frac{lmv(X)}{TMV(DB)} \quad (6)$$

For example, $SH(\{BC\}) = 21 / 58 = 0.36$.

Definition 8 : The minimum local measure value min_lmv is denoted as,

$$min\_lmv = \text{ceiling}(min\_share \times TMV(DB)). \quad (7)$$

For instance, $min\_lmv = ceiling(0.25x58) = 15$. For any patternset X, check whether $lmv(X)$ is less than min_lmv. If $lmv(x)$ satisfies the condition, the patternset X is a share-frequent pattern.

From the property of the downward closure (also called *anti-monotonicity*) of the support measure, it can be seen that this property can maintain in the share-frequent patterns mining by using the *tmv* value of patternset X (Definition 4). For any patternset X, if $tmv(X)$ is less than min_lmv, all supersets of X can be pruned (including X) immediately without further consideration. For example, let minimum share threshold is 0.25, then $tmv(\{A\}) = 12 < min\_lmv = 15$. According to the property of downward closure, all supersets of {A} are not generated as a candidate patterns. Therefore, we can prune {A} at the early step.

III. PROPOSED SOLUTION

The new solution employs PSTable to maintain all non-redundant pattern sets with count information. This is to avoid multiple database scans of mining algorithms and to provide an incremental knowledge source. In addition, it proposes a new mining algorithm called HCG (High Share Candidate Generation) to mine share-frequent patterns from PSTable. The new algorithm reaches its efficiency by generating candidates from only high-share atomic items.

A. Maintenance of PSTable

The PSTable is a set of patterns and their count information. It is an improved version of a BitTable structure [15-16]. It saves space by aggregating transactions with common patternsets. To maintain PSTable, each transaction in original database or a new transaction, treated as a patternset, is checked for its existence in PSTable. If it exists, the maintenance algorithm updates count information of the patternset. If it is a new one, it is inserted into PSTable with its count information. Figure 1 shows the maintenance algorithm in details. Table 2 shows the PSTable constructed from the transaction database in Table 1.

```
1 Input : Transactions of DB or db+, the PSTable
2 Output : the updated PSTable
3 Procedure :
4 For each transaction t∈ DB or db+
5     Check a patternset of t with the existing patternsets in the PSTable
6     if patternset of t∈ the PSTable
7         Increment the total of quantity of transaction t in a total count
8         Increment count of patterns in the pattern count
9     Else
10        Insert the patternset with the total of quantity into the PSTable
          and set the pattern count = 1
11    End If
12End For
```

Figure 1. An algorithm for maintenance the PSTable

TABLE II. THE PS TABLE IS CREATED AFTER A SINGLE SCAN OF THE DATABASE.

PID	A	B	C	D	E	F	G	H	Total Count	Pattern count
1	X	X	X	X			X	X	6	1
2						X		X	7	1
3		X	X	X					16	2
4			X		X				8	2
5		X		X					5	1
6		X	X	X	X				10	1
7	X					X	X		6	1

B. *The HCG algorithm*

The HCG algorithm is to discover share-frequent patterns from the PSTable by a generate-and-test approach. It minimizes number of candidates by generating them from high share *1*-patterns, patterns of length 1 promising to be part of expected share-frequent patterns. Its correctness is supported by the downward closure property. The algorithm in detail is as described below.

Input : The PSTable as shown in Table 2 and *min_share* (δ) = 0.25

Output : Share-frequent patterns

Step 1 : Determination of the *TMV(DB)* = 58 and the *min_lmv*(=*TMV* * δ) =15.

Step 2 : Each *1*-pattern *X* is loaded into the algorithm and calculates its the sum of the total count of all transactions containing *X* (*tmv*(*X*)) as illustrated in Fig. 2. If each *tmv*(*X*) is greater than or equal to *min_lmv*, the *1*-pattern *X* will be kept into a set of high share-pattern *HS*. The algorithm does as the same step until all *1*-pattern is done. Therefore, the pattern in the *HS* is {B}, {C}, {D} and {E}.

Step 3 : The algorithm reads each *1*-pattern *X* in *HS* and then shows all patterns that *X* appears in Table 2. For example, the pattern "B" has four patterns in Fig. 3 (a). Then, the algorithm computes the sum of total count of each *1*-pattern (*tmv*(*X*)).

Step 4 : Check whether the *tmv*(*X*) is larger than or equal to the *min_lmv*. From Fig. 3 (a), the *1*-patterns {B}, {C} and {D} exceed *min_lmv* then this pattern is added to HS_X

Step 5 : Mining all patterns in HS_X that prefix with *X*. From Fig. 3(a), the prefixed pattern "B" generates {B}, {B,C}, {B,D} and {B,C,D} respectively. That is HS_B={{B}, {B,C}, {B,D}, {B,C,D}}. The share-frequent patterns mined from HS_B are {B,C}, {B,D}, and {B,C,D}.

Step 6 : Remove *X* from *HS*. Repeat Steps 3 – 5 until *HS* is empty. From Fig. 3(b), 3(c), and 3(d), we get HS_C={{C}, {C,D}, {C,E}, {C,D,E}}, HS_D={{D}}, and HS_E={{E}}, respectively. The share-frequent patterns mined from HS_C, HS_D, and HS_E are {C} and {C,D}. Table 3 shows all generated candidates and their information.

1-patterns	*tmv*(*i*)
A	12
B	37
C	40
D	37
E	18
F	13
G	12
H	13

Figure 2. The tmv of each 1-pattern is calculated.

PatternID	B	C	D	E	Total Count
1	x	x	x		6
3	x	x	x		16
5	x		x		5
6	x	x	x	x	10
tmv(1-patterns)	**37**	**32**	**37**	10	

a) The process of pattern "B"

PatternID	D	E	Total Count
1	x		6
3	x		16
5	x		5
6	x	x	10
tmv(1-patterns)	**37**	10	

c) The process of pattern "D"

PatternID	C	D	E	Total Count
1	x	x		6
3	x	x		16
4	x		x	8
6	x	x	x	10
tmv(1-patterns)	**40**	**32**	**18**	

b) The process of pattern "C"

PatternID	E	Total Count
4	x	8
6	x	10
tmv(1-patterns)	**18**	

d) The process of pattern "E"

Figure 3. The process of each high share-pattern

TABLE III. THE CANDIDATE PATTERNS GENERATED BY HCG ALGORITHM.

No.	Candidate patterns	tmv	lmv	SH	Share-frequent patterns
1	B	37	14	0.2414	NO
2	**B,C**	**32**	**21**	**0.3621**	**YES**
3	**B,D**	**32**	**22**	**0.3793**	**YES**
4	**B,C,D**	**32**	**27**	**0.4655**	**YES**
5	**C**	**40**	**16**	**0.2759**	**YES**
6	**C,D**	**32**	**16**	**0.2759**	**YES**
7	C,E	18	14	0.2414	NO
8	C,D,E	10	7	0.1207	NO
9	D	37	8	0.1379	NO
10	E	18	4	0.069	NO

IV. RESULT AND DISCUSSION

Experiments were made to compare performance of the proposed algorithm with three algorithms: ShFSM[13], DCG[14], and MCShFI[16]. These algorithms are implemented in Microsoft C# 2012 and executed on a PC with 3.30 GHz CPU and 8GB main memory. The same datasets in [16] are chosen for evaluating the performance of the proposed algorithm. All datasets can be accessed at the FIMI repository page (http://fimi.cs.helsinki.fi/data). The characteristics of these dataset are as shown in Table 4.

TABLE IV. THE CHARACTERISTICS OF ALL DATASETS USED FOR EXPERIMENT EVALUATIONS.

Datasets	#Patterns	#Records	Avg.length
Mushroom	119	8,124	23
T10I4D100K	870	100,000	11

Figure 4 shows execution times (in second) of the proposed algorithm compared with of the other three algorithms on the mushroom dataset under different minimum share thresholds. The mushroom dataset contains 8,124 transactions with 119 different patterns. The average length of the transaction is 23 and around 20% of its patterns appear in every transaction and

it is a dense dataset. On the mushroom dataset, all algorithms are very time consuming for lower minimum share thresholds because there are too many long candidate patterns to generate and test. However, the proposed algorithm takes the lowest run times in all minimum share thresholds. Figure 5 presents number of generated candidates by tested algorithms under different minimum share thresholds. The results confirm that the new algorithm outperforms the previous algorithms on mushroom dataset.

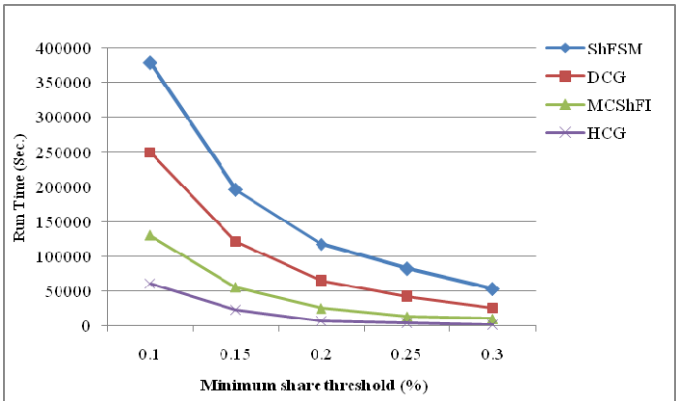


Figure 4. The total processing time comparisons on the mushroom dataset

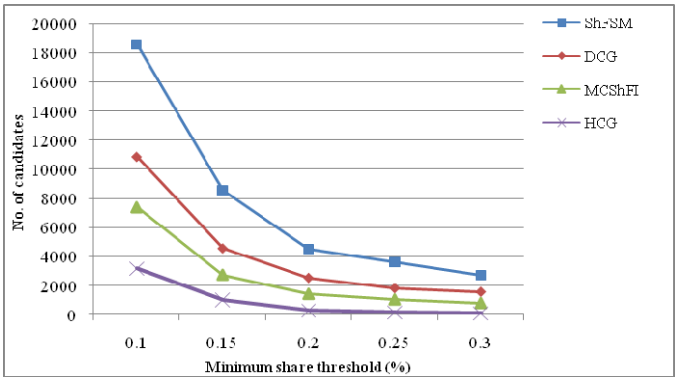


Figure 5. The number of candidate patterns comparison on the mushroom dataset

Next experiments are to evaluate performance of the new algorithm over the T10I4D100K dataset with the other three algorithms under different minimum share thresholds. The T10I4D100K is a simulated dataset generated by the IBM simulator and there are 100,000 transactions with 870 different patterns. The average length of the transaction is 11 and around 1.16% of its patterns appear in every transaction and it is a sparse dataset. Fig. 6 and Fig 7 show comparison of the total processing time (in second) and the amount of generated candidates for four algorithms respectively. It is clearly that the proposed algorithm is better than the existing algorithms because the new algorithm reduces the useless candidate patterns in early steps and only generates the share-frequent patterns from high share different items. As shown in the both Figures that there is no such a pattern that meet the share-frequent patterns when the minimum share threshold is bigger than or equal to 0.07%. Therefore, they show the computation time and the number of candidates that were used initially.

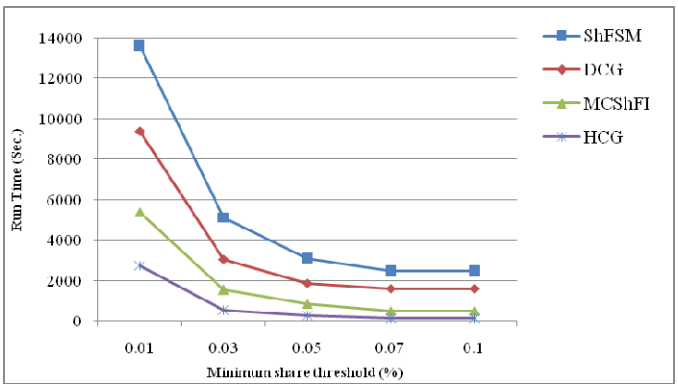


Figure 6. The total processing time comparisons on the T10I4D100K dataset

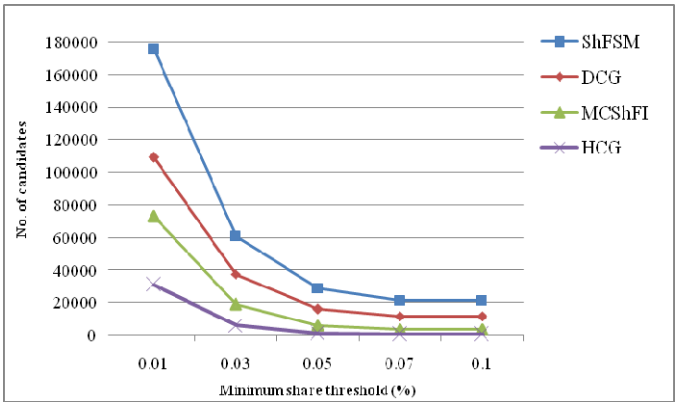


Figure 7. The number of candidate patterns comparison on the T10I4D100K dataset

V. CONCLUSION

The paper proposes to use an incremental pattern set table knowledge (called PSTable) for mining share-frequent patterns. The PSTable for a transaction database is constructed by only a single database scan. Information from a new transaction added to the database can be incrementally added to the PSTable without rescanning the whole database. In addition, the paper proposes a new efficient algorithm named the HCG algorithm to discover share-frequent patterns from the PSTable. With the downward closure property, the new algorithm reaches its efficiency by generating candidates from only high share atomic patterns. The experimental results on dense and sparse datasets show that the proposed algorithm outperforms existing algorithms in both running time and the number of candidates. Moreover, running time of the proposed algorithm could be improved if the PSTable is implemented using a fast access data structure.

REFERENCES

[1] R. Agrawal, T.Imielinski and A.Swami, "Mining association rules between sets of items in large databases", In *Proc. ACM SIGMOD*, 1993, 207-216.

[2] R. Agrawal and R.Srikant, “Fast Algorithms for Mining Association Rules in Large Databases”, In *Proc. VLDB*, Chile, 1994, 487–499.

[3] Jong Soo Park, Ming-Syan Chen and Philip S.Yu, “Using a hash-based method with transaction trimming for mining association rules”, In *TKDE*, VOL.9, NO.5, 1997, 813-825.

[4] S. Brin, R. Motwani, J.D. Ullman and S. Tsur, “Dynamic itemset counting and implication rules for market basket data”, In *Proc. ACM SIGMOD*, May 1997, 255-264.

[5] J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, In *Proc. ACM SIGMOD*, May 2000, 1-12.

[6] R. C. Agarwal, C. C. Aggarwal and V. V. V. Prasad, “A tree projection algorithm for generation of frequent itemsets*”*, In *JPDC*, 2001, 350-371.

[7] J. Pei, J. Han and H. Lu, “Hmine: Hyper-structure mining of frequent patterns in large databases”, In *Proc. IEEE ICDM*, 2001, 441–448.

[8] El-Hajj M and Zaiane OR, “COFI Approach for Mining Frequent Itemsets Revisited”, In *Proc. ACM SIGMOD*, Paris, France, 2004, 70 – 75.

[9] C. L. Carter, H. J. Hamilton and N. Cercone, “Share based measures for itemsets”, In *Principles of Data Mining and Knowledge Discovery. Springer*, Trondheim, Norway, 1997, 14-24.

[10] B.Barber and H. J. Hamilton, “Extracting share frequent itemset with infrequent subsets”, In *Data mining and Knowledge Discovery*, April 2003, 153-185.

[11] B. Barber and H. J. Hamilton, “Algorithms for mining share frequent itemsets containing infrequent subsets”, In *Proc. PKDD*, Lyon, France, 2000, 316-324.

[12] Y.-C. Li, J.-S. Yeh and C.-C. Chang, “Efficient Algorithms for Mining Share-Frequent Itemsets”, In *Proc. IFSA*, 2005, 543 – 539.

[13] Y.-C. Li, J.-S. Yeh and C.-C. Chang, “A fast algorithm for mining share-frequent itemsets”, In *Proc. APWEB*, 2005, 417-428.

[14] Y.-C. Li, J.-S. Yeh and C.-C. Chang, “Direct candidates generation: a novel algorithm for discovering complete share-frequent itemsets”, In *LNCS*, VOL.3614, 2005, 551.

[15] C. Nawapornanan and V. Boonjing, “A new share frequent itemsets mining using incremental BitTable knowledge”, In *Proc. ICCIT*, 2011, pp. 358-362.

[16] C. Nawapornanan and V. Boonjing, “An efficient algorithm for mining complete share-frequent itemsets using BitTable and heuristics”, In *Proc. ICMLC*, 2012, 96-101.

Author Biography

Name	Miss Chayanan Nawapornanan
Date of Birth	01 May 1982
Address	60/97 Moo 15 Tumbon Rachatewa, Amphur Bangphi, Province Samutprakarn, 10540
Education	(2004) Bachelor of Information Technology GPA 3.14 Burapha University
	(2006) Master of Science in Information Technology GPA 3.71 Burapha University
	(2012) Master of Science in Computer Science GPA 4.00 King Mongkut's Institute of Technology Ladkrabang
	(2018) Doctor of Philosophy in Computer Science King Mongkut's Institute of Technology Ladkrabang
Scholarship	Scholarships of Faculty of Science, King Mongkut's Institute of Technology Ladkrabang

Academic Publications

1. HCG: A new algorithm for mining share-frequent patterns (Conference Publication)
2. High Candidates Generation: A New Efficient Method for Mining Share-Frequent Patterns (Journal Publication)
3. An Incremental Approach to Share Frequent Itemsets Mining (Journal Publication)