PAPER Special Section on Knowledge, Information and Creativity Support System

Handling Dynamic Weights in Weighted Frequent Pattern Mining

Chowdhury Farhan AHMED[†a)], **Syed Khairuzzaman TANBEER**[†b)], ***Nonmembers***,
Byeong-Soo JEONG[†*c)], ***Member***, ***and*** **Young-Koo LEE**[†d)], ***Nonmember***

SUMMARY Even though weighted frequent pattern (WFP) mining is more effective than traditional frequent pattern mining because it can consider different semantic significances (weights) of items, existing WFP algorithms assume that each item has a fixed weight. But in real world scenarios, the weight (price or significance) of an item can vary with time. Reflecting these changes in item weight is necessary in several mining applications, such as retail market data analysis and web click stream analysis. In this paper, we introduce the concept of a dynamic weight for each item, and propose an algorithm, DWFPM (dynamic weighted frequent pattern mining), that makes use of this concept. Our algorithm can address situations where the weight (price or significance) of an item varies dynamically. It exploits a pattern growth mining technique to avoid the level-wise candidate set generation-and-test methodology. Furthermore, it requires only one database scan, so it is eligible for use in stream data mining. An extensive performance analysis shows that our algorithm is efficient and scalable for WFP mining using dynamic weights.

key words: *data mining, knowledge discovery, weighted frequent pattern mining, dynamic weight*

1. Introduction

In practice, the frequency of a pattern may not be a sufficient indicator for finding the meaningful patterns in a large transaction database because the frequency only reflects the number of transactions in the database which contain that pattern. In many cases, the items in a transaction can have different degrees of importance (weight). For example, in retail applications, an expensive product may contribute a large portion of overall revenue even though it does not appear in many transactions. For this reason, WFP mining [3]–[8], [28], [29] was proposed as a way to discover more useful knowledge by considering the different weights of each item. The weight-based pattern mining approach can be applied in many areas, such as market data analysis where the prices of products are important factors, web traversal pattern mining where each web page has a different strength of impact, and biomedical data analysis where most diseases are not caused by a single gene but by a combination of genes.

Manuscript received April 9, 2008.
Manuscript revised July 8, 2008.

[†]The authors are with the Department of Computer Engineering, Kyung Hee University, 1 Sochun-ri, Kihung-eup, Youngin-si, Kyonggi-do, 446–701, South Korea.
[*]Corresponding author
a) E-mail: farhan@khu.ac.kr
b) E-mail: tanbeer@khu.ac.kr
c) E-mail: jeong@khu.ac.kr
d) E-mail: yklee@khu.ac.kr

DOI: 10.1093/ietisy/e91–d.11.2578

Even though WFP mining can consider diverse application-specific weights during the mining process, it still cannot reflect a real world environment where the significance (weight) of an item varies with time. Most of the existing WFP [3]–[8], [28] mining algorithms are based on static databases. Zhang et al. [29] proposed a strategy to maintain association rules in dynamic databases by weighting. However, they considered one weight for a dataset containing a group of transactions. By doing so, they highlighted recently added groups of transactions over the previously added groups. This assumption does not hold in the realistic situation where the importance of each item or itemset can vary with time.

In real world scenarios, the significance of each item might be affected by many factors. Peoples' buying behaviors change with time, and affect the significance of products in retail markets. For example, in one period of time, jeans may be the favorite item, but in some other period of time the users' preferred clothes may be trousers and T-shirts. The popularity of seasonal products also varies when the season changes. Consider the situation when the season changes from winter to summer. In this case, the importance of electric fans increases rapidly. On the other hand, the importance of jackets and other warm clothes decreases sharply. The opposite situation occurs when the season changes from summer to winter. Web click stream analysis is another example. The significance of each web site may change dynamically depending on its popularity, political issues, public events, and so on.

Motivated by these real world scenarios, we propose a new strategy for handling dynamic weights in WFP mining. As in the business market the weight (importance or price) of an item may vary due to environmental changes over different time periods, in this paper, we introduce the concept of applying a dynamic weight to every item included in the WFP mining process. Our approach keeps track of the varying weights of each item batch-by-batch in a prefix-tree. To handle the dynamic weights during the mining process, we propose a new algorithm called DWFPM (dynamic weighted frequent pattern mining). Our algorithm can handle dynamically changing item weights. It exploits the pattern growth mining technique to remove the level-wise candidate set generation-and-test methodology of the existing algorithm [29]. Furthermore, it requires only one database scan, so it is eligible for stream data mining [13]–[16]. Our tree-structure achieves larger prefix-sharing and is more space-efficient than the existing algorithm [29]. Extensive performance analyses show that our algorithm is efficient and scalable for dynamic weighted frequent pattern mining, and outperforms the standard algorithm [29].

The remainder of this paper is organized as follows. In Sect. 2, we describe the background of the field. In Sect. 3, we explain our proposed DWFPM algorithm for weighted frequent pattern mining using dynamic weights. In Sect. 4, our experimental results are presented and analyzed. In Sect. 5, we discuss the contribution of our proposed algorithm in practical application areas. Finally, in Sect. 6, our conclusions are presented.

2. Background

2.1 Frequent Pattern Mining

Let $I = \{i_1, i_2, \ldots\ldots i_m\}$ be a set of items, and let D be a transaction database, $\{T_1, T_2, \ldots\ldots T_n\}$, where each transaction, $T_i \in D$, is a subset of I. The support/frequency of a pattern, $X\{x_1, x_2, \ldots\ldots. x_p\}$, is the number of transactions containing the pattern in the transaction database. The goal of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The *downward closure* property [1], [2] is used to prune the infrequent patterns. This property says that if a pattern is infrequent, then all of its super-patterns must be infrequent. The *Apriori* [1], [2] algorithm is the initial solution of the frequent pattern mining problem, but it suffers from the candidate generation and test problem and requires several database scans. During the first database scan, it finds all of the 1-element frequent itemsets, and based on its findings, generates the candidates for the 2-element frequent itemsets. During the second database scan, it finds all of the 2-element frequent itemsets, and based on these findings, generates the candidates for the 3-element frequent itemsets, and so on. FP-growth [9] solves this problem by using an FP-tree-based solution without any candidate generation and using only two database scans. There has been a significant amount of research into finding frequent patterns [9]–[13], [15], [19], [30].

An incremental updating technique called FUP [20] was proposed for the maintenance of the association rules. This work was based on the level-wise candidate set generation-and-test methodology of the *Apriori* algorithm. There has been some approaches [17]–[19], [30] into the single pass mining of traditional frequent pattern mining using the pattern growth approach. This research has shown that incremental prefix-tree structures are efficient using currently available memory sizes (gigabyte-range). Some other single pass research [13]–[16] has been done to find frequent patterns in a data stream in real time. This traditional frequent pattern mining approach considers equal profits/weights for all items.

Table 1 An example of retail database.

Item	Price ($)	Support (frequency)	Normalized Weight
Personal computer	800	500	0.8
Laser printer	450	320	0.45
Bubble jet printer	250	450	0.25
Digital Camera	600	700	0.6
Memory stick	200	825	0.2
Hard disk	130	350	0.13
DVD drive	100	450	0.1
CD drive	50	250	0.05

2.2 Weighted Frequent Pattern Mining

The weight of an item is a non-negative real number which is assigned to reflect the importance of each item in the transaction database. For a set of items, $I = \{i_1, i_2, \ldots\ldots i_n\}$, the weight of a pattern, $P\{x_1, x_2, \ldots\ldots. x_m\}$, is given as follows:

$$Weight(P) = \frac{\sum_{q=1}^{length(P)} Weight(x_q)}{length(P)} \qquad (1)$$

A weighted support of a pattern is defined as the value that results from multiplying the pattern's support with the weight of the pattern. So the weighted support of a pattern, P, is given as follows:

$$Wsupport(P) = Weight(P) \times Support(P) \qquad (2)$$

A pattern is called a weighted frequent pattern if the weighted support of the pattern is greater than or equal to the minimum threshold.

Table 1 shows an example of a retail database in which normalized weight values are assigned to items based on their prices. A normalization process is required to adjust the differences between data from various sources to create a common basis for comparison [3]–[5]. According to the normalization process, the final item weights can be determined to be within a specific weight range. For example, in Table 1 the weight values of the items are in the range from 0.05 to 0.8.

Some weighted frequent pattern mining algorithms (MINWAL [6], WARM [7], WAR [8]) have been developed based on the *Apriori* algorithm using the candidate generation and test paradigm. Obviously, these algorithms require multiple database scans and result in poor mining performance.

WFIM [3] is the first FP-tree-based weighted frequent pattern algorithm using two database scans over a static database. It makes use of a minimum weight and a weight range. Items are assigned fixed weights randomly from within the weight range. The FP-tree is arranged in weight ascending order and maintains the *downward closure* property. WCloset [28] is proposed for the calculation of the closed weighted frequent patterns.

To extract more interesting weighted frequent patterns, the WIP [5] algorithm introduces the new parameter of weight-confidence to measure the strong weight affinity of a pattern. WIP also uses another parameter called hyperclique-confidence [21] to measure the strong support affinity of the weighted patterns. WFIM and WIP use different pruning conditions to find interesting patterns, but their overall mining procedures are almost the same. This means that both of them require two database scans which are not suitable for either stream data mining or incremental/interactive mining.

The WFIM [3] and WIP [5] algorithms show that the main challenge of weighted frequent pattern mining is that the weighted frequency of an itemset (or a pattern) does not have the *downward closure* property. Consider that item "*a*" has weight 0.6 and frequency 4, item "*b*" has weight 0.2 and frequency 5, and itemset "*ab*" has frequency 3. According to Eq. (1), the weight of itemset "*ab*" will be (0.6 + 0.2)/2 = 0.4, and according to Eq. (2) its weighted frequency will be 0.4 × 3 = 1.2. The weighted frequency of "*a*" is 0.6 × 4 = 2.4, and of "*b*" is 0.2 × 5 = 1.0. If the minimum threshold is 1.2, then pattern "*b*" is weighted infrequent but "*ab*" is weighted frequent. WFIM and WIP maintain the *downward closure* property by multiplying each itemset's frequency by the global maximum weight. In the above example, if item "*a*" has the maximum weight of 0.6, then by multiplying it with the frequency of item "*b*", 3.0 is obtained. So, pattern "*b*" is not pruned at this early stage, and pattern "*ab*" will not be missed. At the final stage, this overestimated pattern "*b*" will finally be pruned by using its actual weighted frequency.

Zhang et al. [29] proposed a new strategy for maintaining the association rules in dynamic databases using the weighting technique to highlight new data. Any recently added transactions are assigned higher weights. Moreover, all transactions in a group of databases are given the same weight. For example, in dataset D, all transactions have the same weight, w_1; in dataset D_1^+, all transactions have the same weight, w_2, and so on. They did not use different weights for individual items or transactions. Their algorithm is based on the level-wise candidate set generation-and-test methodology of the *Apriori* algorithm. Therefore, for a particular dataset, they generate a large number of candidates and need to perform several database scans to get the final result.

In the existing weighted frequent pattern mining approaches, no one has proposed a method of dynamic weighted frequent pattern mining in which the weight of an item may be changed in any batch of transactions. Moreover, the existing approaches require multiple database scans. Therefore, we propose an algorithm for dynamic weighted frequent pattern mining using a single database scan.

3. DWFPM: Our Proposed Algorithm

3.1 Definitions

Definition 1: Dynamic weighted support of a pattern, P, is defined by

Table 2 An example of transaction database with dynamic weights.

Batch	TID	Trans.	Weight				
	T_1	a, b, d	a	b	c	d	e
1st	T_2	c, d	0.45	0.9	0.2	0.3	0.5
	T_3	a, b					
	T_4	b	a	b	c	d	e
2nd	T_5	b, c, d	0.6	0.7	0.4	0.5	0.4
	T_6	c, e					
	T_7	a, c, e	a	b	c	d	e
3rd	T_8	a	0.5	0.3	0.7	0.4	0.45
	T_9	a, c					

$$DWsupport(P) = \sum_{j=1}^{N} Weight_j(P) \times Support_j(P) \tag{3}$$

Here N is the number of batches, $Weight_j(P)$ and $Support_j(P)$ are the weight and support of pattern P respectively in the j^{th} batch. We can calculate the value of $Weight_j(P)$ by using Eq. (1). For example, the *DWsupport* of pattern "*bd*" in the first, second and third batches are ((0.9 + 0.3)/2) × 1 = 0.6, ((0.7 + 0.5) / 2) × 1 = 0.6 and ((0.3 + 0.4) / 2) × 0 = 0, respectively in Table 2. So, the total *DWsupport*("*bd*") = 0.6 + 0.6 + 0 = 1.2.

Definition 2: A pattern is called a dynamic weighted frequent pattern if the dynamic weighted support of the pattern is greater than or equal to the minimum threshold. For example, if the minimum threshold is 1.2, then "*bd*" is a dynamic weighted frequent pattern in Table 2.

3.2 Tree Construction

In this section, we describe the construction process of our tree structure which captures transactions having items with dynamic weights. A header table is maintained in our tree as in an FP-tree [9]. The first value in the header table is the item id. After that, the frequency and weight information associated with an item is kept in a batch-by-batch fashion within the header table. The tree nodes only contain an item's id and its batch-by-batch frequency information. To facilitate the tree traversals, adjacent links are maintained in our tree structure as in an FP-tree (not shown in the figures for simplicity).

Consider the example database shown in Table 2. After reading a transaction from the database, first the items inside it are sorted according to lexicographical order, and then they are inserted into the tree. Figure 1 (a) shows the tree after capturing the transactions from batch 1. After the first batch, separate transaction count information must be kept for each batch in both the header table and tree, while weight information is only kept inside the header table. Figure 1 (b) shows the tree after inserting the first and second batches. Because frequency information is kept separately in each node of the tree, we can easily discover which transactions have occurred in which batch. For example, from Fig. 1 (b) we can easily determine that the batch numbers of transactions "*c, d*" and "*c, e*" are 1 and 2 respectively. Figure 1 (c) shows the tree after inserting the first, second, and

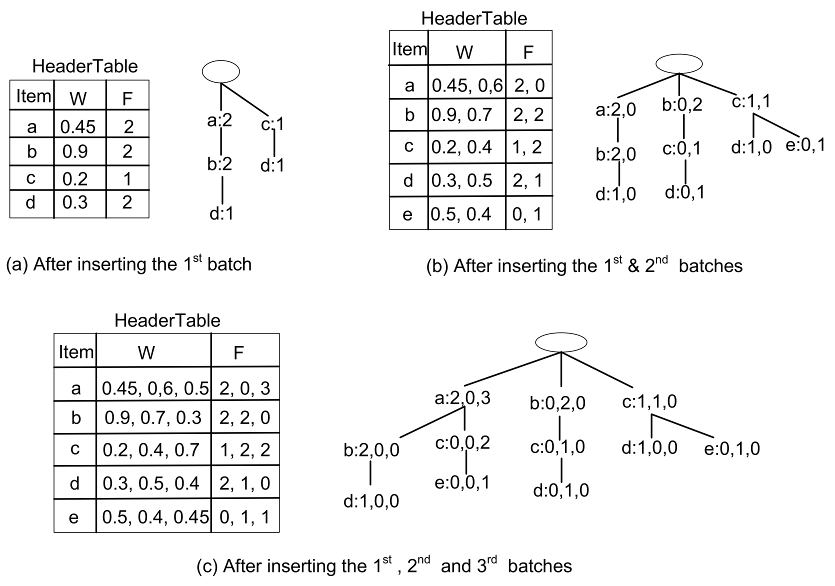

Fig. 1 Tree construction.

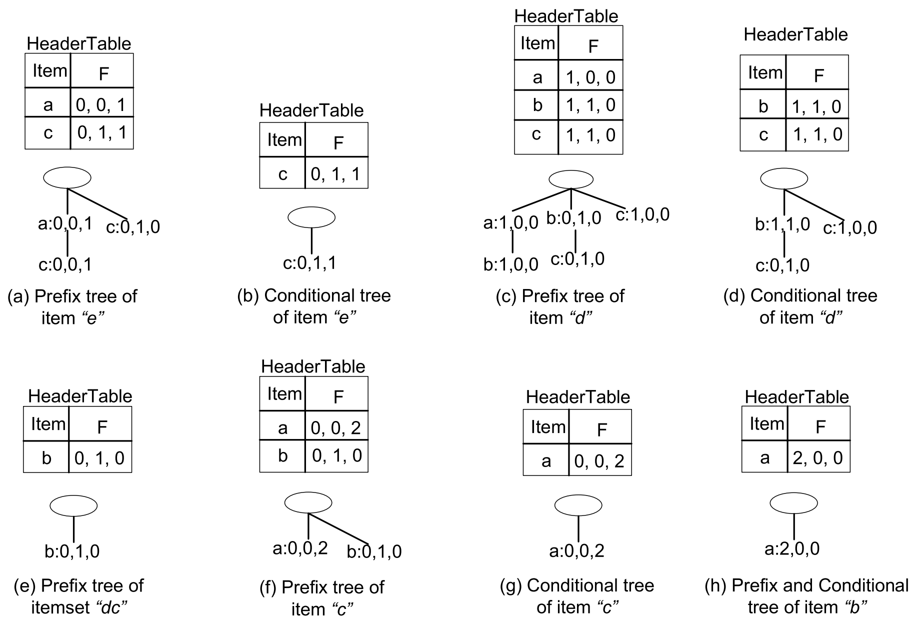

Fig. 2 Mining operation.

third batches. Our tree structure has the following properties:

Property 1: The ordering of items in the tree is not affected by the changes in frequency of the items.

Property 2: The total frequency count at any node in the tree is greater than or equal to the sum of total frequency counts of its children.

Property 3: The tree structure can be constructed in a single database pass.

3.3 Mining Process

In the FP-growth mining algorithm [9], when a prefix tree is created for a particular item, then all branches prefixing that item are taken with the frequency value of that item. After that, the conditional tree is created from the prefix tree by deleting all of the nodes containing infrequent items. DWFPM performs the same type of mining operation. As discussed in Sect. 2.2, the main challenge in this area is that the weighted frequency of an itemset does not have the *downward closure* property, so to maintain this property we have to use the global maximum weight. The global max-

Table 3 *DWsupport* calculations of the candidate patterns.

No.	Candidate patterns	*DWsupport* calculation	Result
1	*ce*: 0,1,1	(((0.4+0.4)/2)×1) + (((0.7+0.45)/2)×1)= 0.4+0.575=0.975	Pruned
2	*e*: 0,1,1	0.4×1+0.45×1 = 0.85	Pruned
3	*cd*: 1,1,0	(((0.2+0.3)/2)×1) + (((0.4+0.5)/2)×1) = 0.25+0.45=0.7	Pruned
4	*bd*: 1,1,0	(((0.9+0.3)/2)×1) + (((0.7+0.5)/2)×1) = 0.6+0.6=1.2	Pass
5	*d*: 2,1,0	0.3×2+0.5×1 = 1.1	Pruned
6	*ac*: 0,0,2	((0.5+0.7)/2))×2 = 1.2	Pass
7	*c*: 1,2,2	0.2×1+0.4×2+0.7×2 = 2.4	Pass
8	*ab*: 2,0,0	((0.9+0.45)/2))×2 = 1.35	Pass
9	*b*: 2,2,0	0.9×2+0.7×2 = 3.2	Pass
10	*a*: 2,0,3	0.45×2+0.5×3 = 2.4	Pass

imum weight is the maximum weight of all of the items in the global database. In our case, this is the highest weight among every weight in all of the batches. For example, in Table 2, item "*b*" has the global maximum weight of 0.9. We will refer this term as *GMAXW*. The local maximum weight is needed when we are doing the mining operation for a particular item, and is not always equal to *GMAXW*. For example, in the database shown in Table 2, item "*e*" does not occur with item "*b*" and "*d*". As a result, in the mining operation, the prefix tree of "*e*" only contains items "*a*" and "*c*". Here we do not need to use *GMAXW* because the local maximum weight can maintain the *downward closure* property. The local maximum weight for "*e*" is the maximum weight of the items "*c*", "*a*", and "*e*", which is 0.7. We will refer to this local maximum weight as *LMAXW*. Using *LMAXW* in place of *GMAXW* reduces a pattern's probability of being a candidate.

Consider the database presented in Table 2, the tree constructed for that database in Fig. 1 (c), and its minimum threshold = 1.2. Here *GMAXW* = 0.9. After multiplying the total frequency of each item with *GMAXW*, the dynamic weighted frequency list is <*a*:4.5, *b*:3.6, *c*:4.5, *d*:2.7, *e*:1.8>. As a result, all items are single element candidates. Now we will construct the prefix and conditional trees for these items in a bottom up fashion, and mine the dynamic weighted frequent patterns. First the prefix tree of the bottom-most item "*e*" is created by taking all of the branches prefixing item "*e*" shown in Fig. 2 (a). To create the conditional tree for this item, we have to delete the nodes from its prefix tree which cannot be candidate patterns. For item "*e*", *LMAXW* = 0.7. After multiplying the frequencies in the header table with *LMAXW* in Fig. 2 (a), we get the dynamic weighted frequency list <*a*:0.7, *c*:1.4>. As item "*a*" has a low dynamic weighted frequency with item "*e*", it must be deleted to get the conditional tree of "*e*". Figure 2 (b) shows the conditional tree of item "*e*". The candidate patterns "*ce*" and "*e*" are generated at this point. The prefix tree of item "*d*" is created in Fig. 2 (c). *LMAXW* = 0.9 here, and the dynamic weighted frequency list is <*a*:0.9, *b*:1.8, *c*:1.8>. By deleting item "*a*" from the prefix tree, we get the conditional tree of item "*d*" (shown in Fig. 2 (d)). The candidate patterns "*bd*", "*cd*", and "*d*" are generated here. The prefix tree of pattern "*dc*" is shown in Fig. 2 (e). The dynamic weighted frequency list is <*b*:0.9>. As a result, no conditional tree or candidate pattern is generated here.

```
Input: DB contains batches of transactions with dynamic
       weights, minimum threshold ($\delta$)
Output: Dynamic weighted frequent patterns

 1. Begin
 2. For each transaction $T_i$ in DB
 3.     Sort the items inside $T_i$ in lexicographical order
 4.     Insert $T_i$ into Tree
 5.     Update the header table H
 6. End For
 7. GMAXW is the global maximum weight
 8. For each item $\alpha_i$ from the bottom of H
 9.     If (totalFreq($\alpha_i$)×GMAX W) > $\delta$ then
10.         Create Prefix tree $PT_i$ for item $\alpha_i$
11.         Calculate LMAXW
12.         Call Mining ($PT_i$, $\alpha_i$, LMAXW)
13.     End If
14. End For
15. End

Procedure Mining (T, $\alpha$, LMAXW)

 1. Begin
 2. Create conditional tree CDT of $\alpha$ by deleting each
        item $d_i$ from T having totalFreq($d_i$)×LMAXW < $\delta$
 3. For each item $\beta_i$ in the header table of CDT
 4.     Call Test_Candidate($\alpha\beta_i$)
 5.     Create Prefix tree $T_\beta$ for itemset $\alpha\beta_i$
 6.     Calculate LMAXW
 7.     Call Mining ( $T_\beta$, $\alpha\beta_i$, LMAXW)
 8. End For
 9. End

Procedure Test_Candidate (X )

 1. Begin
 2. Let Dynamic weighted support of X be $DW_X$
 3. Set $DW_X = 0$
 4. For each batch $B_i$
 5.     $DW_X = DW_X$ + (frequency($X_{Bi}$)×Weight($X_{Bi}$))
 6. End For
 7. If $DW_X \geq \delta$ then
 8.     Add X in the Dynamic weighted frequent pattern list
 9. End If
10. End
```

Fig. 3 The DWFPM algorithm.

The prefix tree of item "*c*" is created in Fig. 2 (f). *LMAXW* = 0.9 here, and the dynamic weighted frequency list is <*a*:1.8, *b*:0.9>. By deleting item "*b*" from the prefix tree, we get the conditional tree of item "*c*" (shown in Fig. 2 (g)). The candidate patterns "*ac*" and "*c*" are generated here. The prefix tree of item "*b*" is created in Fig. 2 (h). *LMAXW* = 0.9, and the dynamic weighted frequency list is <*a*:1.8>. As a result, it is also the conditional tree for item "*b*", and patterns "*ab*" and "*b*" are generated here. The last candidate pattern "*a*" is generated for the top-most item "*a*". We have to test all of the candidate patterns with their actual dynamic weighted frequencies using Eq. (3), and mine the actual dynamic weighted frequent patterns. Table 3 shows these calculations. The actual dynamic weighted frequent patterns are "*bd*":1.2, "*ac*":1.2, "*c*":2.4, "*ab*":1.35, "*b*":3.2, and "*a*":2.4. Finally, pseudo-code of the DWFPM algorithm is presented in Fig. 3.

4. Experimental Results and Analysis

4.1 Experimental Environment and Datasets

To evaluate the performance of our proposed algorithm, we have performed several experiments on IBM synthetic datasets (*T10I4D100K*, *T40I10D100K*) using synthetic weights [22], real life datasets (*retail*, *chess*, *mushroom*, *kosarak*) using synthetic weights [22], [23], and a real dataset (*Chain-store*) using real weight values [24]. The performance of our algorithm was compared with the existing algorithm, "*Weight*" [29], with respect to execution time and memory usage. Our programs were written in Microsoft Visual C++ 6.0, and run in a time sharing environment with the Windows XP operating system on a Pentium dual core 2.13 GHz CPU with 1 GB main memory.

4.2 Performance Study on Execution Time

4.2.1 Synthetic Datasets with Synthetic Weights

In this section we used IBM synthetic datasets *T10I4D100K* and *T40I10D100K* developed by the IBM Almaden Quest research group and obtained from the frequent itemset mining dataset repository [22]. These datasets do not provide the weight values of each item. Most of the previous weight-based frequent pattern mining research [3]–[5], [7], [8], [28], [29] generated random numbers for the weight values of each item, but when observing real world datasets, most items are in the low weight range. Therefore, the weight value of each item was heuristically chosen to be between 0.1 and 0.9, and randomly generated using a log-normal distribution. Some other pattern mining research [26], [27] has adopted the same technique. Figure 4 shows the weight distribution of 2000 distinct items using the log-normal distribution.

The *T10I4D100K* dataset contains 100,000 transactions and 870 distinct items. Its mean transaction size is 10.1, and it is a sparse dataset. Around 1.16% ((10.1 / 870) × 100) of its distinct items are present in every transaction, so it has short dynamic weighted frequent patterns. The *T40I10D100K* dataset contains 100,000 transactions

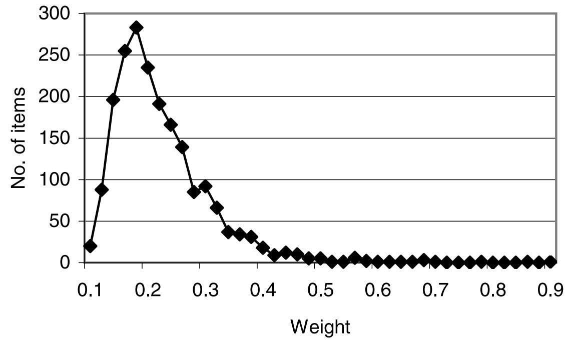

Fig. 4 Weight generation for 2000 distinct items using lognormal distribution.

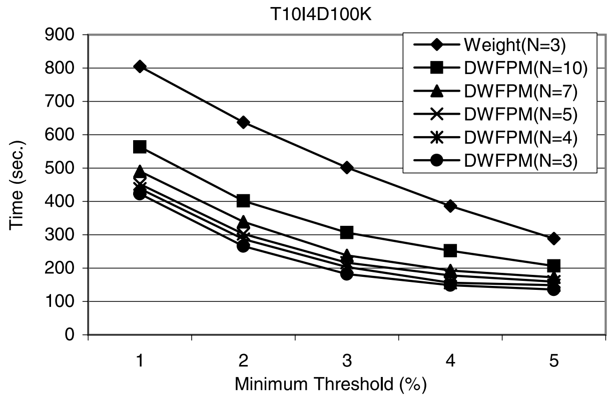

Fig. 5 Execution time on the *T10I4D100K* dataset.

and 942 distinct items. Its mean transaction size is 39.61, and it is a moderately sparse dataset. Around 4.2% ((39.61 / 942) × 100) of its distinct items are present in every transaction, so it has long dynamic weighted frequent patterns compared to *T10I4D100K*.

We have divided these two datasets into 3, 4, 5, 7, and 10 batches, and will use N to represent the number of batches. For N = 4, 5, and 10, all batches contain 25,000, 20,000, and 10,000 transactions, respectively. For N = 3 and 7, all batches contain 35,000, 15,000, and 10,000 transactions, respectively, except the last batch. The numbers of transactions in the last batches are the transactions remaining at the end. For each batch of transactions, we have considered dynamic variation in weights for all items. We have used N = 3 for the existing algorithm "*Weight*". Figure 5 and Fig. 6 show the execution time performance curves for *T10I4D100K* and *T40I10D100K* respectively. The minimum threshold ranges of 1% to 5% and 5% to 25% are used in Fig. 5 and Fig. 6 respectively.

It is obvious from these results that the execution time increases with increasing values of N due to the additional calculations required for each batch. The existing algorithm, "*Weight*", uses the level-wise candidate set generation-and-test approach and therefore generates a large number of candidates. Moreover, for a particular dataset, it must scan the dataset several times. For example, to find all of the weighted frequent patterns from DB_1, it has to scan DB_1

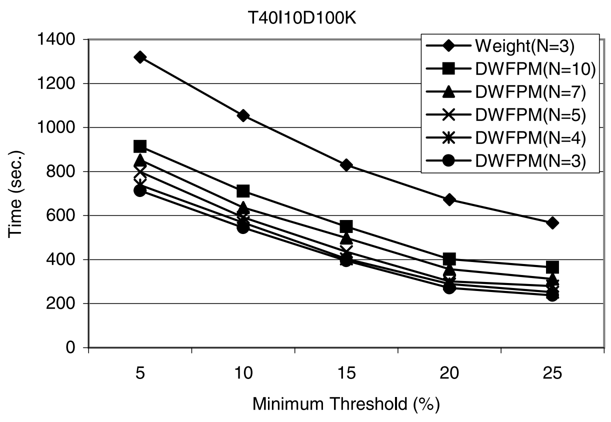

Fig. 6 Execution time on the *T40I10D100K* dataset.

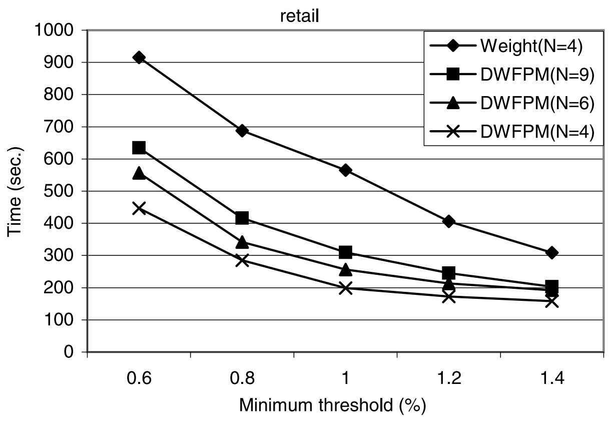

Fig. 7 Execution time on the *retail* dataset.

several times using the level-wise candidate generation-and-test procedure. It must follow the same process for DB_2, DB_3, and so on. In contrast to the existing approach, our algorithm scans each DB_i exactly once, uses an efficient tree structure, and exploits a pattern growth technique to avoid the level-wise candidate set generation-and-test problem. The experimental results in Fig. 5 and Fig. 6 demonstrate that by using an efficient tree structure, the single database scan approach, and the pattern growth mining technique, our algorithm performs better than the existing algorithm.

4.2.2 Real-Life Datasets with Synthetic Weights

In this section, we use real-life datasets (*retail*, *mushroom*, *chess*, and *kosarak*) obtained from the frequent itemset mining dataset repository [22] and the UCI machine learning repository [23]. These datasets do not provide the weight values of each item. As in Sect. 4.2.1, we have generated weights for each item using a lognormal distribution.

The dataset *retail* is provided by Tom Brijs, and contains the retail market basket data from an anonymous Belgian retail store [22], [25]. It contains 88,162 transactions and 16,470 distinct items. Its mean transaction size is 10.3, and it is a large sparse dataset. Around 0.0625%((10.3 / 16470) × 100) of its distinct items are present in every transaction.

We have divided the *retail* dataset into 4, 6, and 9 batches. For $N = 4$, 6, and 9, all of the batches contain 25,000, 15,000, and 10,000 transactions, respectively, except the last batch. For each batch of transactions, we have considered dynamic variation of the weights for all items. We have used $N = 4$ for the existing algorithm, "*Weight*". Figure 7 shows the execution time performance curves for *retail*. The minimum threshold range of 0.6% to 1.4% is used in Fig. 7.

The dataset *mushroom* includes descriptions of hypothetical samples corresponding to 23 species of mushrooms [22], [23]. It contains 8,124 transactions and 119 distinct items. Its mean transaction size is 23, and it is a moderately dense dataset. Almost 20% ((23/119) × 100) of its distinct items are present in every transaction, so it has long

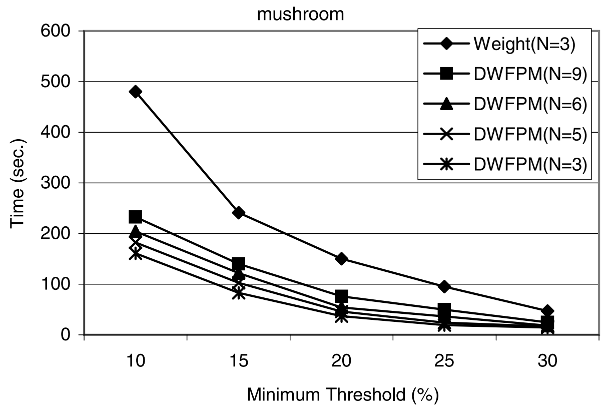

Fig. 8 Execution time on the *mushroom* dataset.

dynamic weighted frequent patterns. We have divided this dataset into 3, 5, 6, and 9 batches. When $N = 3$, the first and second batches contain 3000 transactions, and the third batch contains 2124 transactions. For $N = 5$, 6, and 9, all of the batches contain 2000, 1500, and 1000 transactions, respectively, except the last batch. For each batch of transactions, we have considered dynamic variation of the weights for all items. We have used $N = 3$ for the existing algorithm, "*Weight*". Figure 8 shows the execution time performance curves for *mushroom*. The minimum threshold range of 10% to 30% is used in Fig. 8.

The *chess* dataset was compiled from chess game state information [22], [23]. It contains 3196 transactions and 75 distinct items. Its mean transaction size is 37, and it is a huge dense dataset. Almost 50% ((37/75) × 100) of its total distinct items are present in every transaction, so it has long dynamic weighted frequent patterns. We have divided this dataset into 3, 5, and 6 batches. For $N = 3$, 4, and 5, all of the batches contain 1500, 1000, and 700 transactions, respectively, except the last batch. For each batch of transactions, we have considered dynamic variation of the weights for all items. We have used $N = 3$ for the existing algorithm, "*Weight*". Figure 9 shows the execution time performance curves for *chess*. The minimum threshold range of 50% to 70% is used in Fig. 9.

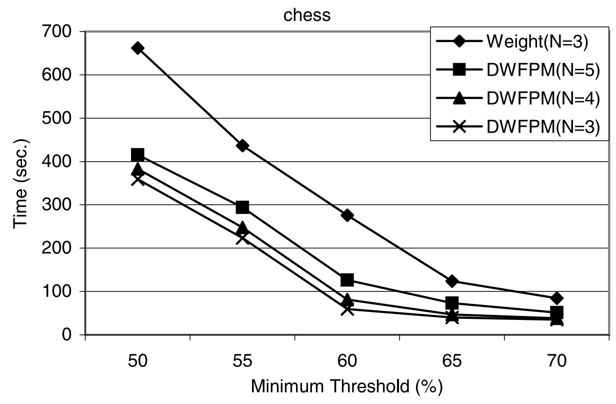

Fig. 9 Execution time on the *chess* dataset.

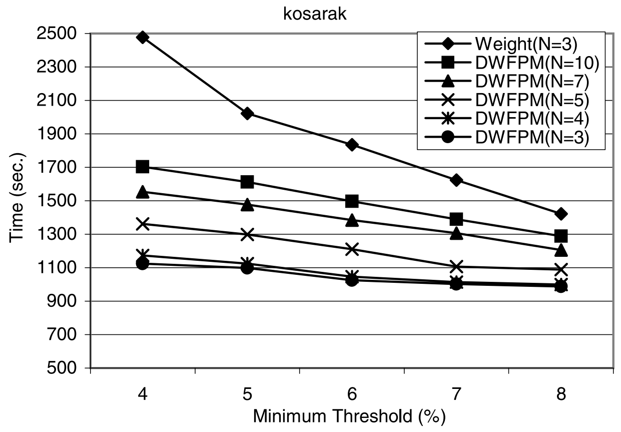

Fig. 10 Execution time on the *kosarak* dataset.

The dataset *kosarak* was provided by Ferenc Bodon and contains click-stream data of a Hungarian on-line news portal [22]. It contains 990,002 transactions and 41,270 distinct items. Its mean transaction size is 8.1, and it is a large sparse dataset. Around 0.0196% ((8.1/41270) × 100) of its distinct items are present in every transaction, so it has too many short dynamic weighted frequent patterns. We have divided this dataset into 3, 4, 5, 7, and 10 batches. For N = 3, 4, 5, 7, and 10, all of the batches contain 350,000, 250,000, 200,000, 150,000, and 100,000 transactions, respectively, except the last. For each batch of transactions, we have considered dynamic variation of the weights for all items. We have used $N = 3$ for the existing algorithm, "*Weight*". Figure 10 shows the execution time performance curves for *kosarak*. The minimum threshold range of 4% to 8% is used in Fig. 10.

4.2.3 Real-Life Dataset with Real Weights

In this section, we use a real-life dataset adopted from NU-MineBench 2.0, a powerful benchmark suite consisting of multiple data mining applications and databases [24]. This dataset called *Chain-store* was taken from a major chain in California and contains 1,112,949 transactions and 46,086 distinct items [24], [27]. We have taken real weight values for items from their price table.

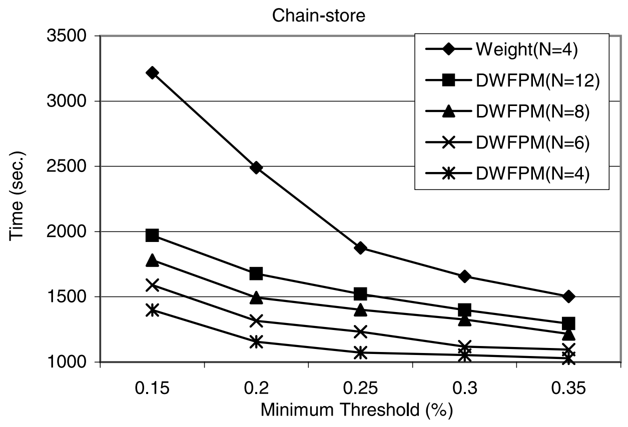

Fig. 11 Execution time on the real-life *Chain-store* dataset.

Table 4 Memory comparison(MB).

Datasets	No. of Batches(N)	DWFPM	*Weight*
T10I4D100K	3	12.72	17.61
T40I10D100K	3	63.51	85.49
retail	4	13.96	21.38
mushroom	3	0.59	1.26
chess	3	0.52	1.13
kosarak	3	196.35	348.67
Chain-store	4	244.61	423.15

The mean transaction size of this real-dataset is 7.2, and it is a large sparse dataset. Around 0.0156% ((7.2 / 46086) × 100) of its distinct items are present in every transaction, so it has too many short dynamic weighted frequent patterns. We have divided this dataset into 4, 6, 8, and 12 batches. For N = 4, 6, 8, and 12, all of the batches contain 300,000, 200,000, 150,000, and 100,000 transactions, respectively, except the last. We have used $N = 4$ for the existing algorithm, "*Weight*". Figure 11 shows the execution time performance curves for this dataset. The minimum threshold range of 0.15% to 0.35% is used in Fig. 11.

The experimental results on real datasets with real and synthetic weights reflect the execution time analyses done in Sect. 4.2.1 for synthetic datasets with synthetic weights. Our algorithm, DWFPM, outperforms the existing algorithm, "*Weight*", by using a single database scan, an efficient tree structure, and the pattern growth mining technique.

4.3 Scalability of DWFPM

It is shown in Fig. 11 that our algorithm has easily handled the 46,086 distinct items and more than 1 million transactions in the real-life *Chain-store* dataset. Figure 10 also shows that it has efficiently handled the 41,270 distinct items and around 1 million transactions in the *kosarak* dataset. Therefore, these experimental results demonstrate the scalability of our algorithm to handle large number of distinct items and transactions.

4.4 Memory Usage

Research into prefix-tree-based frequent pattern mining [17]–[19], [30] has shown that the memory requirement for the prefix trees is low enough to use the gigabyte-range memory now available. We have also handled our tree very efficiently and kept it within this memory range. Our prefix-tree structure can represent transaction information in a very compressed form because transactions have many items in common. By utilizing this type of path overlapping (prefix sharing), our tree structure can save on memory space. Table 4 shows that by using our prefix-tree structure, our proposed algorithm requires much less memory space than the existing algorithm, "*Weight*".

5. Discussion

In this section, we discuss additional practical applications of the proposed approach. In a stock market, the importance of a share can change rapidly over time due to the economic conditions of a country and national or international affairs. Therefore, the unstable values of importance of the different share patterns can be represented by our proposed dynamic weights. By finding dynamic weighted frequent share patterns over a desired time period, stock investors can obtain useful information. In a similar way, our approach can be effective in extracting important knowledge from the auction market in which buyers enter competitive bids and sellers enter competitive offers simultaneously, as opposed to the over-the-counter market, where trades are negotiated.

In application domains such as financial data analysis, the telecommunications industry, and the retail industry, weighted frequent pattern mining can be used to detect unusual access patterns or sequences related to financial crimes, fraudulent telecommunications activities, and the purchase of many expensive items within a short time [28]. In this case, higher weights are given to items which have been previously found in fraudulent patterns [28]. To find these important patterns using static weights, prior information must be given about the items. Our approach of dynamic weighted frequent pattern mining solves this problem by considering weight variation for different items in each batch of transactions.

The importance of a website may change dynamically. For example, during any big international football tournament, the popularity of football-related websites may increase rapidly. These changes in web click stream databases can be handled by our dynamic weight concept. Dynamic weighted frequent pattern mining is also useful for biological gene databases, as each type of gene has a specific importance which changes for different drug analyses.

Global Positioning System (GPS) of Telematics can be found another important application area of weighted frequent pattern mining. In WIP [5], one possible application was the determination of a traffic pattern (a set of links) that considers speed and traffic volume using the weight and frequency information of each link. Candidate weighted frequent patterns (the combination of links) can be calculated according to a user's request to find a path between two locations. The research work WIP [5] used static weights for this real-life application. However, it can be better handled by our proposed dynamic weighted frequent patterns. The speed and traffic volume of each link may vary dynamically over time, so considering the dynamic weight of each link instead of a static weight can provide a more accurate combination of links.

The discussion presented in this paper shows that although there have been some efforts in mining weighted frequent patterns, they have not been suitable for handling real-world scenarios when the importance of a pattern varies dynamically over time. Our proposed dynamic weighted frequent pattern mining approach can effectively handle that situation.

6. Conclusions

The purposes of this paper are to introduce the concept of a dynamic weight for each item in weighted frequent pattern mining, and to provide a new tree-based algorithm to efficiently mine dynamic weighted frequent patterns. By storing batch-by-batch frequency and weight information, our algorithm, DWFPM, discovers accurate knowledge about dynamic weighted frequent patterns. It can mine dynamic weighted frequent patterns with an easy-to-construct tree structure and a fixed sort order. It is also applicable to real time data processing because it requires only one database scan. Our algorithm exploits a pattern growth mining technique to avoid the level-wise candidate generation-and-test problem, and by using an efficient tree structure, can save memory space. Extensive performance analyses show that our algorithm is efficient when applied to both dense and sparse datasets, and can handle a large number of distinct items and transactions.

Acknowledgements

We would like to express our deep gratitude to the anonymous reviewers of this paper. Their useful comments have played a significant role in improving the quality of this work. We are also grateful to professor Yu-Chiang Li [27] for his help in finding the real-life *Chain-store* dataset with real weight values in NU-MineBench 2.0 [24].

This research was supported by a grant of the Korea Health 21 R&D Project, Ministry for Health Welfare and Family Affairs, Republic of Korea (A020602).

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," Proc. 12th ACM SIGMOD Int. Conf. on Management of Data, pp.207–216, May 1993.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," Proc. 20th Int. Conf. on Very Large Data Bases, pp.487–499, Sept. 1994.

[3] U. Yun and J.J. Leggett, "WFIM: Weighted frequent itemset mining with a weight range and a minimum weight," Proc. Fourth SIAM Int. Conf. on Data Mining, pp.636–640, USA, 2005.
[4] U. Yun and J.J. Leggett, "WLPMiner: Weighted frequent pattern mining with length decreasing support constraints," Proc. 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD), pp.555–567, Vietnam, 2005.
[5] U. Yun, "Efficient mining of weighted interesting patterns with a strong weight and/or support affinity," Inf. Sci., vol.177, pp.3477–3499, 2007.
[6] C.H. Cai, A.W. Fu, C.H. Cheng, and W.W. Kwong, "Mining association rules with weighted items," Proc. Int. Database Engineering and Applications Symposium, IDEAS 98, pp.68–77, Cardiff, Wales, UK, 1998.
[7] F. Tao, "Weighted association rule mining using weighted support and significant framework," Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp.661–666, USA, 2003.
[8] W. Wang, J. Yang, and P.S. Yu, "WAR: Weighted association rules for item intensities," Knowledge Information and Systems, vol.6, pp.203–229, 2004.
[9] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," Data Mining and Knowledge Discovery, vol.8, pp.53–87, 2004.
[10] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-Trees," IEEE Trans. Knowl. Data Eng., vol.17, no.10, pp.1347–1362, Oct. 2005.
[11] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions," Data Mining and Knowledge Discovery, vol.15, pp.55–86, 2007.
[12] J. Wang, T. Fukasawa, S. Urabe, T. Takta, and M. Miyazaki, "Mining frequent patterns securely in distributed systems," IEICE Trans. Inf. & Syst., vol.E89-D, no.11, pp.2739–2747, Nov. 2006.
[13] C. Raissi, P. Poncelet, and M. Teisseire, "Towards a new approach for mining frequent itemsets on data stream," Journal of Intelligent Information Systems, vol.28, pp.23–36, 2007.
[14] A. Metwally, D. Agrawal, and A.E. Abbadi, "An integrated efficient solution for computing frequent and top-k elements in data streams," ACM Trans. Database Systems (TODS), vol.31, no.3, pp.1095–1133, 2006.
[15] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining," SIGMOD Record, vol.35, no.1, pp.14–19, March 2006.
[16] C.K.-S. Leung and Q.I. Khan, "DSTree: A tree structure for the mining of frequent sets from data streams," Proc. 6th Int. Conf. on Data Mining (ICDM'06), pp.928–932, 2006.
[17] J.-L. Koh and S.-F. Shieh, "An efficient approach for maintaining association rules based on adjusting FP-tree structures," Proc. DASFAA'04, pp.417–424, 2004.
[18] X. Li, Z.-H. Deng, and S. Tang, "A fast algorithm for maintenance of association rules in incremental databases," Advanced Data Mining and Applications (ADMA 06), vol.4093, pp.56–63, July 2006.
[19] C.K.-S. Leung, Q.I. Khan, Z. Li, and T. Hoque, "CanTree: A canonical-order tree for incremental frequent-pattern mining," Knowledge and Information Systems, vol.11, no.3, pp.287–311, 2007.
[20] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong, "Maintenance of discovered association rules in large databases: An incremental updating technique," Proc. 12th International Conference on Data Engineering, pp.106–114, 1996.
[21] H. Xiong, P.-N. Tan, and V. Kumar, "Hyperclique pattern discovery," Data Mining and Knowledge Discovery, vol.13, pp.219–242, 2006.
[22] Frequent itemset mining dataset repository. Available from http://fimi.cs.helsinki.fi/data/
[23] UCI machine learning repository. Available from http://kdd.ics.uci.edu/
[24] J. Pisharath, Y. Liu, J. Parhi, W.-K. Liao, A. Choudhary, and G. Memik, NU-MineBench version 2.0 source code and datasets. Available from: http://cucis.ece.northwestern.edu /projects/DMS/MineBench.html
[25] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets, "Using association rules for product assortment decisions: A case study," Proc. Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.254–260, 1999.
[26] Y. Liu, W.-K. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," Proc. 1st International Conf. on Utility-Based Data Mining, pp.90–99, Aug. 2005.
[27] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," Data & Knowledge Engineering, vol.64, pp.198–217, 2008.
[28] U. Yun, "Mining lossless closed frequent patterns with weight constraints," Knowledge-Based Systems, vol.210, pp.86–97, 2007.
[29] S. Zhang, C. Zhang, and X. Yan, "Post-mining: Maintenance of association rules by weighting," Information Systems, vol.28, pp.691–707, 2003.
[30] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "CP-tree: A tree structure for single pass frequent pattern mining," Proc. 12th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD), pp.1022–1027, 2008.

Chowdhury Farhan Ahmed received his B.S. and M.S. degrees in Computer Science from the University of Dhaka, Bangladesh in 2000 and 2002 respectively. From 2003–2004 he worked as a faculty member at the Institute of Information Technology, University of Dhaka, Bangladesh. In 2004, he became a faculty member in the Department of Computer Science and Engineering, University of Dhaka, Bangladesh. Currently, he is pursuing his Ph.D. degree in the Department of Computer Engineering at Kyung Hee University, South Korea. His research interests are in the areas of data mining and knowledge discovery.

Syed Khairuzzaman Tanbeer received his B.S. degree in Applied Physics and Electronics, and his M.S. degree in Computer Science from the University of Dhaka, Bangladesh in 1996 and 1998 respectively. Since 1999, he has been working as a faculty member in the Department of Computer Science and Information Technology at the Islamic University of Technology, Dhaka, Bangladesh. Currently, he is pursuing his Ph.D. in the Department of Computer Engineering at Kyung Hee University, South Korea. His research interests include data mining and knowledge engineering.

Byeong-Soo Jeong received his B.S. degree in Computer Engineering from Seoul National University, Korea in 1983, his M.S. degree in Computer Science from the Korea Advanced Institute of Science and Technology, Korea in 1985, and his Ph.D. in Computer Science from the Georgia Institute of Technology, Atlanta, USA in 1995. In 1996, he joined the faculty at Kyung Hee University, Korea where he is now an associate professor at the College of Electronics & Information. From 1985 to 1989, he was on the research staff at Data Communications Corp., Korea. From 2003 to 2004, he was a visiting scholar at the Georgia Institute of Technology, Atlanta. His research interests include database systems, data mining, and mobile computing.

Young-Koo Lee received his B.S., M.S. and Ph.D. in Computer Science from Korea Advanced Institute of Science and Technology, Korea. He is a professor in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests include ubiquitous data management, data mining, and databases. He is a member of the IEEE, the IEEE Computer Society, and the ACM.