Expert Systems with Applications 41 (2014) 3861–3878

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates

Unil Yun [a,*], Heungmo Ryang [a], Keun Ho Ryu [b]

[a] Department of Computer Engineering, Sejong University, Seoul, Republic of Korea
[b] Department of Computer Science, Chungbuk National University, Cheongju, Republic of Korea

ARTICLE INFO

Keywords:
Candidate pruning
Data mining
High utility itemsets
Single-pass tree construction
Utility mining

ABSTRACT

High utility itemset mining considers the importance of items such as profit and item quantities in transactions. Recently, mining high utility itemsets has emerged as one of the most significant research issues due to a huge range of real world applications such as retail market data analysis and stock market prediction. Although many relevant algorithms have been proposed in recent years, they incur the problem of generating a large number of candidate itemsets, which degrade mining performance. In this paper, we propose an algorithm named MU-Growth (Maximum Utility Growth) with two techniques for pruning candidates effectively in mining process. Moreover, we suggest a tree structure, named MIQ-Tree (Maximum Item Quantity Tree), which captures database information with a single-pass. The proposed data structure is restructured for reducing overestimated utilities. Performance evaluation shows that MU-Growth not only decreases the number of candidates but also outperforms state-of-the-art tree-based algorithms with overestimated methods in terms of runtime with a similar memory usage.

1. Introduction

The explosive increase of data gives the motivation to find meaningful knowledge hidden in the huge database, and data mining techniques are used for this purpose. Pattern mining is one of the data mining techniques for discovering useful patterns from the huge databases. Frequent pattern mining (Caldersa, Dextersb, Gillisc, & Goethalsb, 2014; Chuang, Huang, & Chen, 2008; Duonga, Truonga, & Vob, 2014; Lee, Yun, & Ryu, 2014; Pyun, Yun, & Ryu, 2014; Tanbeer, Ahmed, Jeong, & Lee, 2009; Vo, Coenen, & Le, 2013; Wang & Chen, 2011) is a fundamental research topic in the pattern mining area for mining patterns which occur frequently in a transaction database. For frequent pattern mining, Apriori (Agrawal R. 1994) was proposed, and it has been revealed that the algorithm has limitations, multiple database scans and generating a large number of candidate itemsets. To solve this problem, FP-Growth (Han, Pei, & Yin, 2000) was proposed. FP-Growth can discover all frequent patterns with only two database scans. Frequent pattern mining has been applied to different kinds of databases such as sequential databases (Chang, Wang, Yang, Luan, & Tang, 2009), streaming databases (Caldersa et al., 2014), and incremental databases (Leung, Khan, Li, & Hoque, 2007). There are also numerous types of methods according to characteristics of patterns such as closed pattern mining (Chang et al., 2009) and maximal pattern mining (Yun & Ryu 2013). Although frequent pattern mining has played an important role in data mining area, it treats all items with equal importance and represents the occurrence of items in transactions as binary values. Weighted frequent pattern mining (Yun, Lee, & Ryu, 2014) considers the importance of items and share frequent pattern mining (Barber & Hamilton, 2003) represents the occurrence of items in transactions as non-binary values. In these frameworks, patterns with high weight or quantity can be discovered even if they occur infrequently. In the real world area such as market analysis, however, each item not only has relative importance but also is represented as non-binary values in transactions. For example, each item in a supermarket has different price or profit and multiple copies of an item can be sold in a transaction. Hence, the most profitable itemsets cannot be found in those frameworks since profit of an itemset can be calculated by multiplying unit profit of each item in the itemset by the quantities in transactions including the itemset. To find the most valuable itemsets, both the importance and quantity of each item have to be reflected.

In view of this, utility mining (Ahmed, Tanbeer, Jeong, & Choi, 2012; Erwin, Gopalan, & Achuthan, 2008; Hong, Lee, & Wang, 2011; Lee, Park, & Moon, 2013; Lin, Hong, & Lu, 2011; Lin, Lan, & Hong, 2012; Lin, Tu, & Hsueh, 2012; Shie, Hsiao, & Tseng, 2013; Shie, Yu, & Tseng, 2012; Weiss, Zadrozny, & Saar-Tsechansky, 2008; Yeh, Li, & Chang, 2007) has emerged as one of the most significant research issues in pattern mining field. In utility mining, each item

* Corresponding author.
E-mail addresses: yunei@sejong.ac.kr (U. Yun), ryang@sju.ac.kr (H. Ryang), khryu@cbnu.ac.kr (K.H. Ryu).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.11.038

has external utility such as profit or price, and internal utility which indicates non-binary values of items in transactions such as item quantity. Mining high utility itemsets refers to finding all itemsets with high importance, and the importance of an itemset is measured by the concept of utility. Utility of an itemset is defined as the sum of the product of external utility and internal utility of each item in the itemset. If the utility of an itemset is no smaller than a user-specified minimum utility threshold, the itemset is called a high utility itemset. High utility itemset mining reflects real world market data (Ahmed, Tanbeer, Jeong, & Lee, 2009), and thus it can play an important role in market analysis such as finding the most valuable itemsets which contribute to the major part of the total profits in a retail business. Moreover, high utility itemset mining can be used in other application areas such as web click stream analysis, cross-marketing in retail stores (Erwin et al., 2008), mobile commerce environment planning (Shie, Hsiao, Tseng, & Yu, 2011), and even biological gene database analysis (Ahmed et al., 2009). However, mining high utility itemsets is not an easy task because downward closure property (Agrawal & Srikant, 1994) does not hold. In other words, pruning search space is difficult in high utility itemset mining since a superset of a low utility itemset may be a high utility itemset. To address this problem, previous works (Ahmed et al., 2009; Li, Yeh, & Chang, 2008; Liu et al., 2005) employed overestimated methods. In the frameworks with the methods, algorithms satisfy the downward closure property using the sum of utilities of transactions containing each candidate itemset. Although all the high utility itemsets can be found by applying them, these methods often generate a large number of candidates since they add utilities of items that are not included into super itemsets of the candidate to the sum. That is, the algorithms calculate overestimated utilities more than necessary. As a result, the mining performance is degraded. The reason is that more number of generated candidates consumes more execution time. To address the problem, methods were proposed for more reducing the overestimated utilities (Tseng, Shie, Wu, & Yu, 2013; Tseng & Shie, 2010). They compute decreased overestimating utilities with only items that can be contained the super itemsets, which results in reducing search space. In this framework, however, a large number of candidates are still extracted. For these reasons, we need to reduce the number of candidates to improve mining performance in the models. In other words, the first goal of this study is to improve the mining performance in FP-Tree (Han et al., 2000) based high utility itemsets mining through effective candidate itemset pruning. Meanwhile, a task of scanning database can be a high time consuming process especially when the size of the database is significantly huge in the real world. Moreover, data structures constructed with a single-pass can be utilized to discover patterns efficiently from incremental databases. In the real world applications with such databases, there is a need to update data structures using new information without an additional scan for the whole databases in order to perform efficient itemset mining. It signifies that the data structures that are constructed and restructured with a single-pass can be applied for incremental mining. In this paper, we also aim to develop a tree structure created with a single-pass and a restructuring method of the data structure. Furthermore, we suggest a method for reducing overestimated utilities in the restructuring process. In this paper, motivated from the above, we propose an algorithm as well as a tree structure constructed with a single-pass for efficiently discovering high utility itemsets by utilizing methods for decreasing overestimated utilities and pruning candidates in the restructuring and mining processes. Major contributions of this paper are summarized as follows:

(1) A tree structure, named MIQ-Tree (Maximum Item Quantity Tree), for discovering high utility itemsets is proposed. MIQ-Tree can be constructed with a single-pass by using a restructuring method for reducing overestimated utilities. By applying the restructuring method, we can employ more reduced overestimating utilities.
(2) We propose an algorithm, called MU-Growth (Maximum Utility Growth), with two techniques to prune candidates effectively in the mining process. By adopting the proposed algorithm, high utility itemsets can be generated from MIQ-Tree efficiently.
(3) Various experiments on different types of both real and synthetic datasets are performed for performance comparison between the proposed algorithm and state-of-the-art tree-based algorithms (Ahmed et al., 2009; Tseng et al., 2010; Tseng et al., 2013). Experimental results show that MU-Growth outperforms the other algorithms in terms of execution time and the number of candidates with a similar memory usage, especially when databases contain lots of long transactions or low minimum utility thresholds are set.

The remainder of this paper is organized as follows. In Section 2, we introduce the related work and preliminaries. In Section 3, we describe the proposed tree structure and the restructuring process, and the proposed mining algorithm for effectively pruning candidates is illustrated. In Section 4, we show and analyze experimental results for performance evaluation. Finally, discussion and conclusions of this paper are given in Section 5 and 6.

2. Background

2.1. Related work

For mining frequent patterns, many researches (Caldersa et al., 2014; Chuang et al., 2008; Duonga et al., 2014; Tanbeer et al., 2009; Vo et al., 2013; Wang & Chen, 2011) have been extensively studied, and the initial solution is Apriori (Agrawal & Srikant, 1994). FP-Growth (Han et al., 2000) which is based on pattern growth method was afterward proposed to achieve a better performance than Apriori-based methods. In the FP-Growth algorithm, a tree structure, called FP-Tree, is used. To satisfy two opposite types of constraint, anti-monotone and monotone constraints, MFS_DoubleCons (Duonga et al., 2014) was proposed for mining frequent patterns. It generates all frequent patterns quickly and distinctly with the constraints. In the framework of frequent pattern mining, meanwhile, various kinds of databases are also considered such as sequential databases (Chang et al., 2009), incremental databases (Leung et al., 2007), and stream databases (Yun et al., 2014). Max-Freq-Miner (Caldersa et al., 2014) finds frequent itemsets in a data stream by maintaining a very compact summary of the stream. Although frequent pattern mining has played an important role in pattern mining field, the importance of items and item quantities in transactions are not considered in contrast to real world retail databases. Weighted frequent pattern mining (Yun & Ryu, 2013; Yun et al., 2014) emerged to reflect the relative importance of items in databases. In the framework of weighted frequent pattern mining, weight of a pattern is the ratio of the sum of weight values of items in the pattern to its length. WIT-FWI (Vo et al., 2013) is a tree-based algorithm for discovering frequent weighted itemsets with a tree structure, called WIT-Tree, based on the concept of weight, where relative importance of items are considered. In addition, to satisfy downward closure property (Agrawal & Srikant, 1994), overestimated weights are applied in the framework. Nevertheless, the non-binary occurrence of items in transactions is not considered. To address this issue, utility mining (Erwin et al., 2008; Hong et al., 2011; Lin et al., 2011; Lin et al., 2012; Shie et al., 2011; Shie et al., 2013; Weiss et al., 2008; Wu, Lin, Yu, & Tseng, 2013; Wu, Shie, Tseng, & Yu, 2012; Yeh et al., 2007; Yin, Zheng, &

Cao, 2012) has been extensively studied. In the framework of utility mining, maintaining downward closure property is a difficult task, and TWU (Transaction Weighted Utilization) model has defined (Liu & Choudhary 2005), which is an overestimated method. Two-Phase (Liu et al., 2005), which is based on Apriori algorithm, discovers high utility itemsets. However, the Two-Phase demands multiple database scans and generates a huge number of candidate itemsets because of a level-wise method. To reduce candidates, FUM and DCG + algorithms based on IIDS (Isolated Items Discarding Strategy) were proposed (Li et al., 2008). Although they can decrease the number of candidates, they still need multiple database scans and apply a candidate generation-and-test approach. To avoid multiple database scans and generate high utility itemsets efficiently, IHUP (Ahmed et al., 2009) was proposed. It uses three tree structures, $IHUP_L$-Tree, $IHUP_{TF}$-Tree, and $IHUP_{TWU}$-Tree, which are based on FP-Tree. Each node in the trees is composed of an item name, a support count, and a TWU value. IHUP generates all high utility itemsets from the IHUP-Tree through three steps. In the first step, items in transactions are sorted according to lexicographic order, and the transactions are inserted into $IHUP_L$-Tree with a single database scan. For a single-pass tree construction, the constructed tree can be restructured without additional database scan by arranging nodes by support descending order ($IHUP_{TF}$-Tree) or TWU descending order ($IHUP_{TWU}$-Tree). Note that $IHUP_{TWU}$-Tree showed the best performance in their experiments (Ahmed et al., 2009). In the second step, candidate itemsets are extracted from IHUP-Tree by FP-Growth, and actual high utility itemsets are identified with an additional database scan in the last step. Although IHUP can construct a tree and discover high utility itemsets with two database scans, it generates a large number of candidates by apply the TWU model. To address this issue, UP-Growth (Tseng, Wu, Shie, & Yu, 2010) has recently been proposed, and it uses PHU (Potential High Utility) model. For reducing the number of candidate itemsets, the UP-Growth applies four strategies, DGU (Discarding Global Unpromising items), DGN (Decreasing Global Node utilities), DLU (Discarding Local Unpromising items), and DLN (Decreasing Local Node utilities). Besides, it constructs a tree structure, named UP-Tree, with two database scans and conducts mining high utility itemsets. In other words, it demands three database scans for discovering high utility itemsets. In the first database scan, TWU values of each item are accumulated. In the second database scan, items having less TWU values than the user-specified minimum utility threshold are removed from each transaction. In addition, items in transactions are arranged according to TWU descending order and the transactions are inserted into the UP-Tree. In this stage, DGU and DGN are applied for reducing overestimated utilities. After that, high utility itemsets are generated from the UP-Tree with DLU and DLN. To more decrease overestimated utilities in local UP-Trees, UP-Growth+(Tseng, Shie, Wu, & Yu, 2013) was proposed. It stores a minimal utility value to each corresponding node. Moreover, UP-Growth+ reduces the overestimated utilities with DNU (Discarding local unpromising items and their estimated Node Utilities) and DNN (Decreasing local Node utilities for local UP-Tree by estimated utilities of descendant Nodes), which are improved strategies of DLU and DLN. On the other hand, CHUD and DAHU (Wu & Yu, 2011) based on the concept of closed pattern were proposed. CHUD conducts mining Closed+ high utility itemsets based on closed pattern and DAHU recovers all high utility itemsets from the Closed+ high utility itemsets. Although it can perform mining high utility itemsets faster than the UP-Growth by using a method for reducing the number of candidates, it needs much more amount of memory usage since it uses a database converted into a vertical database. Recently, list-based algorithms (Liu & Qu, 2012; Liu et al., 2012) have been proposed for mining high utility itemsets. HUI-Miner (Liu & Qu 2012) discovers high utility itemsets with a list data structure, called utility list. It first creates an initial utility list for itemsets of the length 1 for promising items. Then, HUI-Miner constructs recursively a utility list for each itemset of the length k using a pair of utility lists for itemsets of the length $k-1$. For mining high utility itemsets, each utility list for an itemset keeps the information of TIDs for all of transactions containing the itemset, utility values of the itemset in the transactions, and the sum of utilities of remaining items that can be included to super itemsets of the itemset in the transactions. Meanwhile, the framework of utility mining can be applicable to various environments due to considering the characteristics of real world applications. UMSP (Shie et al., 2013) mines high utility user behavior patterns from mobile commerce environments. In addition, HURM (Lee et al., 2013) conducts utility-based association rule mining for cross-selling in business environments.

As stated above, reducing the number of candidates is an important issue of tree-based algorithms for mining high utility itemsets with overestimated methods. Therefore, this study aims to decrease the number of candidates by reducing overestimated utilities and pruning the candidate itemsets. Furthermore, another goal of this study is to develop a method for pushing an overestimated utility decreasing strategy in the restructuring process with a single-pass tree construction. By applying the proposed tree structure with both a method for restructuring and a mining algorithm, we can construct the tree with a single-pass and highly decrease the number of candidate itemsets.

2.2. Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items and a database $D = \{T_1, T_2, \ldots, T_n\}$ be a set of n transactions. Each transaction T_d $(1 \leqslant d \leqslant n)$ in D has a unique identifier, called *TID*. An itemset X is a set of k distinct items $\{i_1, i_2, \ldots, i_k\}$, where $X \subseteq I$ and $1 \leqslant k \leqslant m$, and k refers to the length of X and *k-itemset* is an itemset with the length k. In utility mining, each item i_p $(1 \leqslant p \leqslant m)$ in a transaction T_d is associated with a quantity, called *internal utility*, $iu(i_p, T_d)$; e.g., $iu(E, T_3) = 3$ in Table 1. Each item i_p has a unit profit, called *external utility*, $eu(i_p)$; e.g., $eu(E) = 2$ in Table 2. Utility of an item i_p in a transaction T_d is denoted as $u(i_p, T_d)$ and defined as the product of internal and external utilities, $eu(i_p) \times iu(i_p, T_d)$, and utility of an itemset X in a transaction T_d containing X is denoted as $u(X, T_d)$ and defined as $\sum u(i_p, T_d)$, where $i_p \in X$ and $X \subseteq T_d$. Also, utility of an itemset X in the database D is represented as $u(X)$ and defined as the sum of utilities of X in all transactions including X, $\sum u(X, T_d)$, where $X \subseteq T_d$ and $T_d \in D$. For an example, let an itemset $X = \{BE\}$ which is a subset of four transactions T_1, T_3, T_5, and T_8 in Table 1. Utility of X in T_1 can be computed as $u(BE, T_1) = u(B, T_1) + u(E, T_1) = (3 \times 2) + (2 \times 4) = 14$. In the same way, $u(BE, T_3) = 15$, $u(BE, T_5) = 8$, and $u(BE, T_8) = 7$, and thus $u(BE) = 44$. *Transaction Utility* (*TU*) of a transaction T_d is denoted as $TU(T_d)$ and defined as the sum of utilities of each item i_p in the transaction T_d, $\sum u(i_p, T_d)$, where $i_p \in T_d$. Therefore, the total utility of the database D indicates the sum of transaction utilities, $\sum TU(T_d)$, where $T_d \in D$ and $1 \leqslant d \leqslant n$. For instance, in Tables 1 and 2, transaction utility of T_3 can be calculated by $TU(T_3) = u(B, T_3) + u(C, T_3) + u(E, T_3) + u(F,$

Table 1
An example database.

TID	Transaction	TU
T_1	(A, 1) (B, 2) (E, 4)	19
T_2	(A, 2) (B, 1) (D, 1)	17
T_3	(B, 3) (C, 2) (E, 3) (F, 4)	25
T_4	(C, 1) (D, 2) (E, 1)	11
T_5	(B, 2) (D, 1) (E, 1) (F, 3)	18
T_6	(C, 3) (E, 2)	7
T_7	(A, 1) (B, 3) (C, 1) (D, 2)	23
T_8	(A, 2) (B, 1) (C, 3) (D, 1) (E, 2)	24

Table 2
Profit table.

Item	A	B	C	D	E	F
Profit	5	3	1	4	2	2

T_3) = 25, and the total utility of D is $\sum TU(T_d)$ = 144. If a user-specified minimum utility threshold *minutil* is given and utility of an itemset X is no less than *minutil*, X is called a *high utility itemset*; otherwise, it is a *low utility itemset*. High utility itemset mining finds all the high utility itemsets from databases. In high utility itemset mining, maintaining the downward closure property is essential, and *Transaction-Weighted Downward Closure* (*TWDC*) is used for this purpose. *Transaction-Weighted Utility* (*TWU*) of an itemset X, *TWU*(X), refers to the sum of transaction utilities of transactions that include X, and it is defined as $\sum TU(T_d)$, where $X \subseteq T_d$ and $T_d \in D$. In the TWDC model, if a TWU value of an itemset X is no less than *minutil*, then the itemset X is a *high transaction-weighted utilization itemset*, and high transaction-weighted utility itemsets are not pruned in mining process. In Tables 1 and 2, for example, if *minutil* is 50.4 (35%), then an itemset {*BE*} is not pruned in advance even though $u(BE) = 44$ because $TWU(BE) = TU(T_1) + TU(T_3) + TU(T_5) + TU(T_8) = 86$.

3. Proposed tree structure and method

In this section, we propose an algorithm, MU-Growth, as well as a tree structure for mining high utility itemsets. The framework of the proposed method consists of three steps. In the first step, the initial tree is constructed using items and quantities associated with the items in transactions, and then the tree is restructured by arranging nodes according to TWU descending order. Moreover, reduced overestimating utilities of the nodes are calculated during the restructuring process with the quantities, not construction process. In the second step, candidate itemsets are generated from the restructured tree by MU-Growth. In the last step, actual high utility itemsets are identified.

3.1. Proposed tree structure: MIQ-Tree

The proposed tree structure, named MIQ-Tree (Maximum Item Quantity Tree), is used to keep the information of transactions and high utility itemsets. Each item quantity in transactions is employed to construct the initial tree. In addition, the stored quantity information in the tree is used to decrease the overestimated utilities and to construct a global tree with a single-pass. In previous tree structures with a single-pass (Ahmed et al., 2009), accumulated TWU values are saved as node utilities. As a result, a large number of candidates are generated in mining process and mining performance is degraded. In the following subsections, the elements of MIQ-Tree are first defined. Next, we illustrate how to construct and restructure the initial tree in detail. Finally, mining algorithm, MU-Growth for reducing the number of candidates is described.

3.1.1. Elements in MIQ-Tree

In MIQ-Tree, each node N is composed of *N.name*, *N.count*, *N.nu*, *N.parent*, *N.nodelink*, and a set of child nodes. Furthermore, in contrast to the previous tree structures with a single-pass (Ahmed et al., 2009), *N.max* is also an element to store information of item quantities for computing reduced overestimating utilities with this information when the initial tree is restructured. MIQ-Tree saves maximum item quantities to the element. In addition, it is not contained in local trees for memory efficiency. *N.name* is the item name of node. *N.count* is the support count of node. *N.nu* is the node utility and stores overestimated utility. *N.parent* and *N.nodelink* point to the parent node of N and a node which has the same item name with *N.name*, respectively. A header table is also included in the MIQ-Tree to facilitate tree traversal in restructuring and mining processes. Each entry in the header table consists of an item name, an overestimated utility, and a link. The link points to the last occurrence of a node that has the same item name. An item support is only contained in local header tables, and it is employed for reducing the number of candidates in mining process. By using the links in the header and the node links, the nodes with the same name can be traversed efficiently.

3.1.2. Construction of MIQ-Tree

We first explain the process for constructing the initial MIQ-Tree. In the first database scan, each transaction is inserted into the initial tree and its TU is calculated; at the same time, TWU of each item in the transaction is also accumulated to its entry. To insert transaction $T_d = \{(i_1, q_1), (i_2, q_2), \ldots, (i_n, q_n)\}$ ($i_k \in I$, $1 \leqslant k \leqslant n$), where i_k is an item and q_k is a quantity associated with i_k, a function *Insert_Transaction*(*Tree*, T) is called. This subroutine is shown in Fig. 1, where *Tree* is the initial tree and T is a transaction. Each item i_k with the quantity q_k, (i_k, q_k), in T_d is added to the tree as a node, N_{ik} such that $N_{ik}.name = i_k$. First, the node N_{i1} for (i_1, q_1) is created or found under the root node N_R. After the creation, when there is no node N_{i1} under N_R, $N_{i1}.count$ and $N_{i1}.sum$ are initialized as zero. Then, $N_{i1}.count$ is increased by 1, and $N_{i1}.max$ is assigned as a maximum value of $N_{i1}.max$ and q_1. After that, N_{i2} for (i_2, q_2) is processed as a child node of N_{i1}, iterating the same steps for the remaining items.

Example 1. Consider the database and the profit table in Tables 1 and 2. To construct the initial MIQ-Tree, the first transaction $T_1 = \{(A, 1), (B, 2), (E, 4)\}$ is inserted to the tree, and the root node N_R is empty at this stage. Initially, N_A is added under N_R as a child node for (A, 1) because N_R is empty at the very beginning. Then, $N_A.count$ and $N_A.max$ are initialized, and $N_A.count$ is increased by 1. Moreover, $N_A.max$ is set to 1. After that, the second item (B, 2) is inserted as a child node of N_A. In here, N_A has no child node, and thus N_B is added under N_A as a child node. Then $N_B.count$ and $N_B.max$ are assigned as 1 and 2, respectively. The last item E in T_1 is also processed in the same way. N_E is added under N_B as a child node, and then $N_E.count$ and $N_E.max$ are set to 1 and 4, respectively. Next, the following transaction $T_2 = \{(A, 2)\ (B, 1)\ (D, 1)\}$ is inserted. The first item (A, 2) is added as a child node of N_R, but a child node for item A is not created since there is a child node N_A for (A, 1) in T_1 under N_R. Hence, $N_A.count$ is raised by 1, and $N_A.max$ is set to the maximum value between $N_A.max$ and 1. The next item (B, 1) is processed without creating a node due to the same reason with (A, 1). Then, $N_B.count$ is increased by 1, and $N_B.max$ is not changed because the value of $N_B.max$ is greater than 1. In contrast to the other items, (A, 2) and (B, 1), N_D is created under N_B since there is no node for D under N_B. After that, $N_D.count$ and $N_D.max$ are assigned as 1. In the same way, the other transactions are inserted to the initial tree.

Subroutine: *Insert_Transaction*(*Tree*, T)

```
01. N ← the root node of Tree
02. For each item (i, q) in T do
03.     If N has not a child node N_i such that N_i.name = i then
04.         Create a new child node N_i under N
05.         Set N_i.name = i, N_i.count = 0, N_i.max = 0
06.     Increase N_i.count by 1
07.     If N_i.max < q then N_i.max = q
08.     N ← N_i
```

Fig. 1. Subroutine of *Insert_Transaction*.

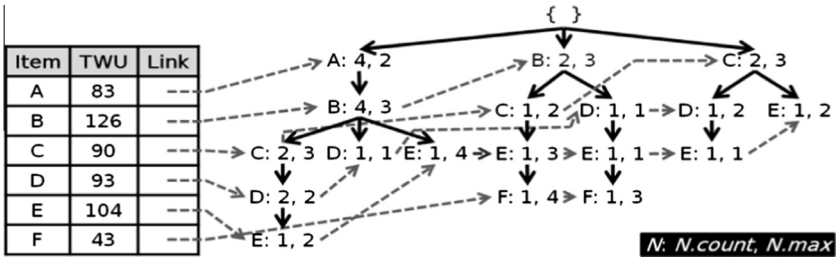


Fig. 2. Constructed MIQ-Tree.

Fig. 2 is the initial MIQ-Tree after inserting transactions in the database of Table 1. The initial tree is restructured with quantity information of nodes to facilitate the performance of utility mining by reducing overestimated utilities when the tree is constructed with a single scan. Initially, each path P is extracted from the tree and reordered according to TWU descending order; at the same time, path utility of each node in P is computed. In here, the path utility refers to reduced overestimating utility. Let $\{N_{i1}, N_{i2}, \ldots, N_{il}\}$ be nodes of a tree T and the nodes be in TWU descending order, where an item i_1 has the highest TWU value. That is, N_{i1} is a child node of the root node and N_{il} is a leaf node. Mining is performed in bottom-up manner, and therefore a conditional pattern tree created from T for an item i_k, $\{i_k\}$-CPT, consists of nodes $\{N_{i1}, N_{i2}, \ldots, N_{ik-1}\}$, where $1 \leqslant k \leqslant l$. That is, an itemset $\{i_1, i_2, \ldots, i_k\}$ is the longest itemset that can be mined from $\{i_k\}$-CPT. Hence, utilities of nodes under N_{ik}, $\{N_{ik+1}, N_{ik+2}, \ldots, N_{il}\}$ in T do not need to calculate overestimated utilities of candidate itemsets from $\{i_k\}$-CPT. In other words, utility of each node in $\{N_{i1}, N_{i2}, \ldots, N_{ik}\}$ can be discarded from path utilities of each node in $\{N_{i1}, N_{i2}, \ldots, N_{ik-1}\}$ (Tseng et al., 2010; Tseng et al., 2013). If there is a path $P = \langle N_A, N_B, N_C\rangle$ in TWU descending order, for example, path utility of N_C is the same with utility of the path P. It means that path utility of the last item in a path is the sum of utilities of all items in the path. That is, the path utility of N_C is the sum of utilities of N_A, N_B, and N_C. On the other hand, path utility of N_B is the sum of utilities of N_A and N_B, and that of N_A is the same with utility of N_A. Utility of a node N_{ik} can be computed by product of external utility of i_k and internal utility of i_k in P. Therefore, the proposed tree structure maintains node quantities for calculating reduced overestimating utilities with a single-pass.

For restructuring the initial tree, each path $P = \langle N_{i1}, N_{i2}, \ldots, N_{im}\rangle$ is extracted from the tree. Here, a path support of P, $sup(P)$ is the difference of a support count of the last node N_{im} and the sum of support counts of child nodes of N_{im} in the tree. For example, a path support of the path $\langle A, B, C, D\rangle$ in Fig. 2 is $sup(\langle N_A, N_B, N_C, N_D\rangle) = 1$. When the path P is extracted, each support count of nodes $\{N_{i1}, N_{i2}, \ldots, N_{im}\}$ in the tree is decreased by $sup(P)$. In this stage, if a support count of a node becomes zero, the node is eliminated from the tree. For instance, when the path $\langle N_A, N_B, N_C, N_D, N_E\rangle$ is extracted from the initial tree of Fig. 2, the node N_E such that $N_E.support = 1$ and $N_E.max = 2$ is removed. Then, nodes in P are rearranged according to TWU descending order, and path utility of each node in the sorted path is calculated with each maximum node quantity and external utility. After that, the sorted path is inserted into the tree with the path utilities.

Definition 1. Let $P = \langle N_{i1}, N_{i2}, \ldots, N_{im}\rangle$ be a path, $P_{sorted} = \langle N'_{i1}, N'_{i2}, \ldots, N'_{im}\rangle$ be a sorted path, and N'_{ik} be a node in P_{sorted}, where $1 \leqslant k \leqslant m$. *Path utility* of N'_{ik} in P_{sorted} is denoted as $pu(N'_{ik}, P_{sorted})$ and defined as $eu(i_k) \times N'_{ik}.max \times sup(P) + pu(N'_{ik-1}, P_{sorted})$.

Especially, path utility of the first node in a sorted path, $pu(N'_{i1}, P_{sorted})$, is $eu(i_1) \times N'_{i1}.max \times sup(P)$ since there is no previous node in the sorted path. Besides, path utility of the next node is $pu(N'_{i2}, P_{sorted}) = eu(i_2) \times N'_{i2}.max \times sup(P) + pu(N'_{i1}, P_{sorted})$. In the same way, path utilities of the rest nodes are computed.

Example 2. Consider the path $P = \langle N_A, N_B, N_C, N_D, N_E\rangle$ in the initial MIQ-Tree of Fig. 2. The last node is N_E, and thus path support of P, $sup(P)$, is 1. First, the path is sorted in TWU descending order. Then, $P_{sorted} = \langle N'_B, N'_E, N'_D, N'_C, N'_A\rangle$, and path utility of N_B, which is the first node of P_{sorted}, is $pu(N'_B, P_{sorted}) = eu(B) \times N'_B.max \times sup(P) = 3 \times 3 \times 1 = 9$, and that of the next node is $pu(N'_E, P_{sorted}) = eu(E) \times N'_E.max \times sup(P) + pu(N'_B, P_{sorted}) = 2 \times 2 \times 1 + 9 = 13$. Path utilities of the other nodes are $pu(N'_D, P_{sorted}) = 21$, $pu(N'_C, P_{sorted}) = 24$, and $pu(N'_A, P_{sorted}) = 34$. After the calculation, P_{sorted} is added to the tree again with the path utilities. Here, each node utility is increased by corresponding path utility. Initially, N_B such that $N_B.count = 2$ and $N_B.max = 3$ under the root node in Fig. 2 is processed because the first node in P_{sorted} is N'_B. $N_B.count$ is raised by $sup(P)$, and $N_B.nu$ is set as $pu(N'_B, P_{sorted})$ since $N_B.nu$ is zero. Then, N_E is created under N_B. The reason is that there is no child node for E under N_B in the initial tree. After that, $N_E.count$ and $N_E.nu$ are assigned as $sup(P)$ and $pu(N'_E, P_{sorted})$. Nodes for the rest, N'_D, N'_C, and N'_A, in P_{sorted} are created in the tree, and then support counts and node utilities are set by following the same steps. Fig. 3 is the result of restructuring the initial tree of Fig. 2.

3.2. Proposed mining algorithm: MU-Growth

In tree-based high utility itemset mining with overestimated methods, reducing the number of candidates is a crucial challenge because identifying actual high utility itemsets from candidates is very time consuming task. It means the more number of candidate itemsets leads to the more execution time, and thus it is need to decrease the number of candidates for efficiently mining high utility itemsets. The proposed algorithm, MU-Growth (Maximal Utility Growth) can reduce the number of candidate itemsets effectively with real item utilities, minimum and maximum item utilities, item utilities, and supports of local items. If estimated maximum utility of a candidate itemset is less than *minutil*, the candidate itemset is pruned. In other words, the candidate itemset is not generated in mining process. In the following subsections, we describe each method of MU-Growth and mining process in detail.

3.2.1. Pruning 1-itemset candidates with real item utilities

The process of mining high utility itemsets is conducted in bottom-up manner. That is, a global MIQ-Tree is traversed by following each link of an item i_g in a header table, which is a global item in the tree. Moreover, a conditional pattern base for each item i_g, $\{i_g\}$-CPB, is constructed from the global tree by extracting all paths starting from nodes for i_g to the root node. In addition, candidate

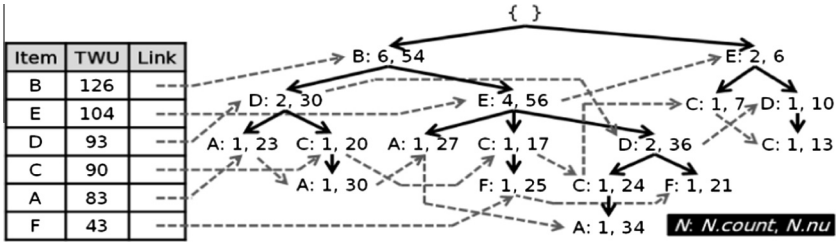


Fig. 3. Restructured MIQ-Tree.

itemsets including the item i_g are generated from a conditional pattern tree for $\{i_g\}$, $\{i_g\}$-CPT, where $TWU(i_g) \geqslant minutil$. In the previous works (Ahmed et al., 2009; Tseng et al., 2010; Tseng et al., 2013), $\{i_g\}$ is generated as a 1-itemset candidate from the conditional pattern tree because i_g has no less TWU value than *minutil*. However, the proposed method prunes 1-itemset candidates using real item utilities of each item in a global tree. We note that each transaction utility is calculated in the first database scan, and thus we can easily obtain real item utilities. Table 3 is the real item utility table for the database in Table 1.

Definition 2. *Real item utility* of an item i_p in a database D is denoted as $u_{Real}(i_p)$ and defined as $\sum u(i_p, T_d)$, where $T_d \in D$ and $i_p \in T_d$.

Example 3. Consider the restructured MIQ-Tree in Fig. 3 and assume that *minutil* is 50.4 (35%). First, tree traversal is performed starting from the bottom-most item F in the header table. In this stage, a conditional pattern base for F is not created due to the less $TWU(F)$ than *minutil*. Thus, it moves the next item A without mining process for F. As a result, any candidate itemset which includes the item F is not generated from the MIQ-Tree. In contrast to F, a conditional pattern base for A, $\{A\}$-CPB, is constructed since the TWU value of A is greater than *minutil*. Then, the process of mining high itemsets for A is conducted with the $\{A\}$-CPB. That is, a candidate itemset which contains the item A can be extracted. However, A is in transactions, T_1, T_2, T_7, and T_8, in the database of Table 1, and $u(A, T_1) = 5$, $u(A, T_2) = 10$, $u(A, T_7) = 5$, and $u(A, T_8) = 10$. Hence, $u_{Real}(A) = 30$ as shown in Table 3. That is, real item utility of A, $u_{Real}(A)$ is 30, and it is less than *minutil*. Thus, 1-itemsset candidate $\{A\}$ is not extracted even though $TWU(C)$ is no less than *minutil*.

3.2.2. Pruning candidates with estimated maximum itemset utility

MU-Growth performs mining high utility itemsets with only items having no less TWU values than *minutil* in a global tree. A conditional pattern base for each item is created; at the same time, supports of items in the conditional pattern base are counted. Then, items that have lower TWU values than *minutil* are pruned unlike to those of a global tree. In addition, a local tree is constructed based on the conditional pattern base. We also use estimated maximum utility of each candidate itemset to eliminate candidates having less estimated maximum utility than *minutil*. Moreover, in construction of local trees, a method for decreasing overestimated utility is employed. In contrast to the method applied to a global tree, it uses minimum item utilities for computing path utility of each item in a path. The reason is that we cannot know actual utility of each item in local trees. To create a local tree, items which have less TWU value than *minutil* are pruned, and then we discard utilities of the removed items from path utilities. In this stage, the discarded utilities of items are calculated by the product of minimum item utility of the each pruned item and support count of the item. For example, if path utility of a path is 10, support of the path is 2, and minimum item utility of a pruned item is 4, then discarded utility of the item is $4 \times 2 = 8$. Then, the utility is discarded from the path, and thus path utility becomes 2.

Definition 3. *Minimum item utility* of an item i_p in a database D is denoted as $u_{Min}(i_p)$ and defined as $MIN(u(i_p, T_d))$, where $T_d \in D$ and $i_p \in T_d$.

Definition 4. *Maximum item utility* of an item i_p in a database D is denoted as $u_{Max}(i_p)$ and defined as $MAX(u(i_p, T_d))$, where $T_d \in D$ and $i_p \in T_d$.

For example, $u_{Min}(E) = 2$ in T_4 and $u_{Max}(E) = 8$ in T_1 in Table 1. Meanwhile, the following definition is used for reducing local estimated utilities based on the DLU strategy (Tseng et al., 2010; Tseng et al., 2013).

Definition 5. Let $CP = \langle i_1, i_2, \ldots, i_k : s, u\rangle$ be a path in a conditional pattern base, where s is support of the path and u is utility of the path without utilities of pruned items. *Local path utility* of an item i_p in CP is denoted as $lpu(i_p, CP)$ and defined as $u - \sum(u_{Min}(i_q) \times s)$, where $1 \leqslant p \leqslant q \leqslant k$.

Each extracted path in a local tree is reordered by TWU descending order, and local path utility of each item in the path is computed, which is similar to reduced path utility in a global tree. Consider the conditional pattern base for C in Table 5. When path utility of each item of the sorted path $\langle B, D\rangle$ is calculated, path utility of D is assigned as utility of the path, 20. This is because the utility of the path includes utilities of all items, B and D, in the path, and the path utility of the last item D also contains them. Therefore, utility of the path and path utility of the last item are the same. In addition, support of the path is 1 and minimum item utility of A is 7 in Table 4. Hence, path utility of B in the path is $20 - (4 \times 1) = 16$. After that, the conditional pattern tree for $\{C\}$ is constructed using paths in Table 5 with the path utilities, and Fig. 4 is the $\{C\}$-CPT.

Table 3
Real item utility table.

Item	A	B	C	D	E	F
Utility	30	36	10	28	26	14

Table 4
Minimum and maximum item utility table.

Item	A	B	C	D	E	F
Min	5	3	1	4	2	6
Max	10	9	3	8	8	8

Table 5
Conditional pattern base for {C}.

Path: path utility	Support
<B, D>: 20	1
<B, E>: 17	1
<B, E, D>: 24	1
<E>: 7	1
<E, D>: 13	1

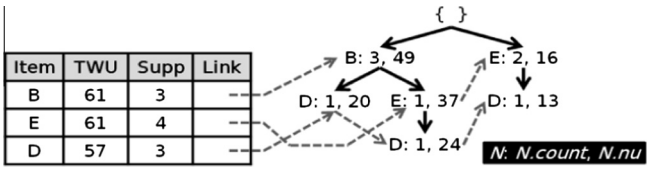


Fig. 4. Conditional Pattern Tree for {C}.

Definition 6. *Global item support* of an item i_g in a database D is denoted as $gsup(i_g)$ and defined as a support count of i_g in a global tree. For example, there are five transactions containing the item C, T_3, T_4, T_6, T_7, and T_8, in D, and thus global item support of C in Table 1 is $gsup(C) = 5$.

Lemma 1. *Consider a candidate itemset $\{i_1, i_2, \ldots, i_k\}$ $(k \geqslant 2)$, which has the length of more than 2, where i_1 is an item selected from a global tree and i_k is the most recently selected item from a local tree for $\{i_1, i_2, \ldots, i_{k-1}\}$. Then, utility of the item i_1 in the candidate itemset is no greater than $u_{Real}(i_1) - (u_{Min}(i_1) \times (gsup(i_1) - sup(i_k)))$.*

Proof. Items, i_1 and i_2, are selected from a global tree and a conditional pattern tree for $\{i_1\}$, respectively. i_3 is selected from a conditional pattern tree for $\{i_1, i_2\}$, and so on. Hence, i_k is the most recently selected item from a conditional pattern tree for $\{i_1, i_2, \ldots, i_{k-1}\}$. If a support count of i_k in the local tree for $\{i_1, i_2, \ldots, i_{k-1}\}$ is $sup(i_k)$, it indicates that there are $sup(i_k)$ transactions containing $\{i_1, i_2, \ldots, i_k\}$ in the database D. In other words, all items in the itemset $\{i_1, i_2, \ldots, i_k\}$ have the same support with $sup(i_k)$ in the tree. On the other hand, $\{i_1, i_2, \ldots, i_{k-1}\}$ is a subset of the candidate itemset $\{i_1, i_2, \ldots, i_k\}$, and thus support of $\{i_1, i_2, \ldots, i_k\}$, $sup(\{i_1, i_2, \ldots, i_k\})$, is no greater than $sup(\{i_1, i_2, \ldots, i_{k-1}\})$ by downward closure property. Moreover, a maximum support of an item i_1 is $gsup(i_1)$. That is, the number of transactions which include i_1 in D is $gsup(i_1)$, and thus $gsup(i_1) \geqslant sup(i_1)$ in the tree. This is because $\{i_1, i_2, \ldots, i_k\}$ is a superset of $\{i_1\}$. Meanwhile, each utility of i_1 in each transaction containing i_1 is no less than $u_{Min}(i_1)$ and the total utility of i_1 in D is $u_{Real}(i_1)$. Consequently, utility of i_1 in $gsup(i_1) - sup(i_k)$ transactions that include i_1 without i_k is no less than $u_{Min}(i_1) \times (gsup(i_1) - sup(i_k))$. Hence, utility of i_1 in $sup(i_k)$ transactions containing $\{i_1, i_2, \ldots, i_k\}$ is no greater than $u_{Real}(i_1) - (u_{Min}(i_1) \times (gsup(i_1) - sup(i_k)))$. □

Definition 7. Consider a conditional pattern tree for $\{i_1, i_2, \ldots, i_k\}$, where i_1 and i_k are selected from a global tree and a conditional pattern tree for $\{i_1, i_2, \ldots, i_{k-1}\}$, respectively. *Estimated maximum global item utility* of i_1 in a candidate itemset $\{i_1, i_2, \ldots, i_k, i_p\}$ from a local tree for $\{i_1, i_2, \ldots, i_k\}$ is denoted as $emgiu(i_1, \{i_1, i_2, \ldots, i_k, i_p\})$ and defined as $u_{Real}(i_1) - (u_{Min}(i_1) \times (gsup(i_1) - sup(i_p)))$.

For example, estimated maximum global item utility of C in a candidate itemset $\{C, D\}$ in Fig. 4 is $emgiu(C,\{C,D\}) = u_{Real}(C) - (u_{Min}(C) \times (gsup(C) - sup(D))) = 10 - (1 \times (5 - 3)) = 8$.

Lemma 2. *Consider the candidate itemset $\{i_1, i_2, \ldots, i_k\}$ $(k \geqslant 2)$ in Lemma 1. Utility of the candidate itemset is no greater than $emgiu(i_1, \{i_1, i_2, \ldots, i_k\}) + \sum(u_{Max}(i_j) \times sup(i_k))$, where $2 \leqslant j \leqslant k$.*

Proof. The number of transactions which include the candidate itemset $\{i_1, i_2, \ldots, i_k\}$ is $sup(i_k)$, and utility of each item i_j, where $2 \leqslant j \leqslant k$, in the each transaction is no greater than $u_{Max}(i_j)$. Therefore, utility of i_j in the candidate itemset in the transactions is less than or equal to $u_{Max}(i_j) \times sup(i_k)$. In addition, utility of i_1 is no greater than $emgiu(i_1, \{i_1, i_2, \ldots, i_k\})$ according to Lemma 1. Hence, utility of the candidate itemset is no greater than to $emgiu(i_1, \{i_1, i_2, \ldots, i_k\}) + \sum(u_{Max}(i_j) \times sup(i_k))$. □

Definition 8. *Estimated maximum item utility* of an item i_p in a candidate itemset $\{i_1, i_2, \ldots, i_k\}$ in a local tree for $\{i_1, i_2, \ldots, i_{k-1}\}$, where $2 \leqslant p \leqslant k$, is denoted as $emiu(i_p, \{i_1, i_2, \ldots, i_k\})$ and defined as $u_{Max}(i_p) \times sup(i_k)$.

For example, estimated maximum item utility of D in a candidate itemset $\{C, D\}$ in Fig. 4 is $emiu(D, \{C, D\}) = u_{Max}(D) \times sup(D) = 27$.

Definition 9. *Estimated maximum candidate itemset utility* of a candidate itemset $\{i_1, i_2, \ldots, i_k\}$ is denoted as $emciu(\{i_1, i_2, \ldots, i_k\})$ and defined as $emgiu(i_1, \{i_1, i_2, \ldots, i_k\}) + \sum emiu(i_p, \{i_1, i_2, \ldots, i_k\})$, where $2 \leqslant p \leqslant k$.

Example 4. Consider the construction of the conditional pattern tree for $\{C\}$ from the global MIQ-Tree of Fig. 3. Table 5 is the conditional pattern base for $\{C\}$, $gsup(C) = 5$, $u_{Real}(C) = 10$, $u_{MIN}(C) = 1$, and *minutil* is set as 50.4 (35%). Moreover, in $\{C\}$-CPT of Fig. 4, $sup(B)$ is 3 and its TWU value is 61. Estimated maximum global item utility of C $emgiu(C, \{C, B\}) = u_{Rreal}(C) - (u_{Min}(C) \times (gsup(C) - sup(B))) = 10 - (1 \times (5 - 3)) = 8$ and $emiu(B, \{C, B\}) = u_{Max}(B) \times sup(B) = 9 \times 3 = 27$. Thus, estimated maximum candidate itemset utility of $\{C, B\}$ is $emciu(\{C, B\}) = 8 + 27 = 35$. As a result, $\{C, B\}$ is not extracted since it has less estimated utility than *minutil*, 50.4. Next, estimated maximum global item utility of C, $emgiu(C, \{C, E\}) = 10 - (1 \times (5 - 4)) = 9$, and $emiu(E, \{C, E\}) = 8 \times 4 = 32$. Hence, $emciu(\{C, E\}) = 9 + 32 = 41$. In the same way, $emciu(\{C, D\}) = 35$. Therefore, all the candidate itemsets composed of two items are not generated from $\{C\}$-CPT in Fig. 4.

Candidate itemsets (1) $\{E, B: 56\}$, and (2) $\{D, B: 66\}$ are generated from conditional pattern trees for $\{E\}$ and $\{D\}$, respectively. In the same way, (3) $\{A, E, B: 61\}$, (4) $\{A, C, B, D: 62\}$, (5) $\{A, D, B: 83\}$, and (6) $\{A, B: 114\}$ are generated from the conditional pattern tree for $\{A\}$. To identify actual high utility itemsets from the candidates, the second database scan is performed, and as a result three high utility itemsets, $\{A, D, B: 56\}$ and $\{A, B: 51\}$ are mined.

Fig. 5 is the proposed algorithm for mining high utility itemsets. Initially, a global MIQ-Tree is traversed in bottom-up manner (line 1). If real item utility of an item i_g having no less TWU value than *minutil* is greater than or equal to the threshold, MU-Growth generates $\{i_g\}$ as a candidate itemset (line 2). After that, node utilities of nodes for i_g in the tree are accumulated, and then a conditional pattern tree for $\{i_g\}$ is created when the computed value is no less than *minutil* (lines 3–7). In mining process of local trees, each estimated maximum itemset utility of a candidate itemset is calculated, and the candidate itemset is extracted when the estimated utility is greater than or equal to *minutil* (line 17). As stated in Fig. 5, MU-Growth algorithm is performed recursively (line 19).

4. Performance evaluation

4.1. Experimental environment and datasets

In this section, experiments for performance evaluation of the compared algorithms: the proposed one and state-of-the-art ones, IHUP (Ahmed et al., 2009), UP-Growth (Tseng et al., 2010), and UP-

MU-Growth(Tree)

/* bottom-up manner */
For each item i_g such that $i_g.twu \geq minutil$ in the header of *Tree* **do**
  /* Pruning 1-itemset candidates with Real Item Utility */
  If $u_{Real}(i_g) \geq minutil$ **then Generate** a candidate itemset $\{i_g\}$
  $twu \leftarrow \phi$
  For each node N_{ig} such that $N_{ig}.name = i_g$ in *Tree* **do**
    Increase *twu* by $N_{ig}.nu$
  If $twu \geq minutil$ **then**
    Create $\{i_g\}$-CPT T_{ig}
    Call *MU-Growth*(T_{ig}, $\{i_g\}$)

MU-Growth(T_X, X)

For each item i such that $i.twu \geq minutil$ in the header of T_X **do**
  $twu \leftarrow \phi$
  For each node N_i such that $N_i.name = i$ in T_X **do**
    Increase *twu* by $N_i.nu$
  If $twu \geq minutil$ **then**
    $X \leftarrow X \cup \{i\}$
    if $N.count = 0$ **then Remove** N from *Tree*
  /* Pruning candidates with Estimated Maximum Itemset Utility */
  If $emciu(X) \geq minutil$ **then Generate** a candidate itemset X
  Create CPT for X, T_X
  Call *MU-Growth*(T_X, X) /* recursive call */

Fig. 5. MU-Growth algorithm.

Growth+(Tseng et al., 2013) are conducted. All the algorithms were written in C++ language. For the evaluation, the experiments were performed on a 3.3 GHz Intel Processor with 8 GB main memory. In addition, the experiments ran with the Windows 7 operating system. Since IHUP algorithm, especially $IHUP_{TWU,}$ based on a single-pass tree construction showed the best performance in Ahmed et al. (2009), we compare ours with $IHUP_{TWU}$.

Both real and synthetic datasets are used in the experiments. Table 6 shows the characteristics of the real datasets. Accidents, Connect, and Retail datasets are acquired from FIMI Repository (http://fimi.cs .Helsinki.fi); Accident dataset contains anonymous traffic accident data, and it is quite dense. Chain-store dataset is obtained from NU-MineBench 2.0 (Pisharath et al., 2005). Foodmart is acquired from Microsoft foodmart 2000 database, and we use foodmart data of Sales Fact 1998. Chain-store and Foodmart already includes external and internal utilities. Connect contains information about online connection. It is a dense dataset. It means that most of the transactions have similar form, and the number of items is small and the average length of the transactions is long. Retail dataset is about product sales data in retail stores, which is a sparse dataset. It has lots of items and the average length is short. In Table 6, $|D|$ is the number of transactions in a dataset, T_{avg} is the average length of transactions, and $|I|$ is the number of items in the dataset. In addition, the characteristics of synthetic datasets are described in Table 7. They are used to evaluate scalability and generated from the data generator (Agrawal & Srikant, 1994). In the datasets T10.I4.DxK from T10.I4.D100K to T10.I4.D1000K, their items are constant, but the number of transactions is gradually increased. In another datasets Ta.Nb.Lc from T10N10000L1000 to T40N40000L4000, the number of items and the average length of transactions are gradually raised. In all of the datasets, except for Chain-store and Foodmart, external utilities for items are generated between 0.01 and 10 by using a log-normal distribution, and internal utilities are generated randomly between 1 and 10 like the previous algorithms (Ahmed et al., 2009; Tseng et al., 2010; Tseng et al., 2013).

Table 6
Characteristics of real datasets.

Dataset	|D|	T_{avg}	|I|	Type
Accidents	340,183	33.8	468	Dense
Chain-store	1,112,949	7.2	46,086	Sparse
Connect	67,557	43	129	Dense
Foodmart	36,869	4.45	1560	Sparse
Retail	88,162	10.3	16,470	Sparse

4.2. Performance comparison with UP-Growth and UP-Growth+

In this part, we compare the performance of MU-Growth with state-of-the-art tree-based algorithms for mining high utility itemsets, UP-Growth (Tseng et al., 2010) and UP-Growth+ (Tseng et al., 2013). Experiments for performance evaluation are conducted as follows. Initially, the algorithms compute a TWU value of each item and sort transactions according to a TWU descending order. Then, they insert the sorted transactions without unpromising items that have smaller TWU values than a minimum utility threshold. After inserting all of the transactions in a database, a global tree is constructed. Next, the algorithms generate candidate itemsets recursively from the tree. In here, pruning techniques are applied. Finally, they identify actual high utility itemsets from the candidates. In order to show effectiveness of our pruning methods, we also conduct experiments for them, and their notations are MU-Strategy 1 (pruning 1-itemset candidates with real item utilities) and 2 (pruning candidates with estimated maximum itemset utility).

4.2.1. Performance comparison on different datasets

We first present performance of the compared algorithms on real utility datasets, Chain-store and Foodmart, in Fig. 6. Fig. 6 (a) and (c) are results in terms of the total runtime and the number of candidates on Chain-store. In Fig. 6 (a), the total runtime of MU-Growth applying two pruning strategies is the best, followed by the first strategy MU-Strategy 1, the second one MU-Strategy 2, and both UP-Growth and UP-Growth+ are the worst. That is, MU-Growth mines all of the high utility itemsets with the fastest speed. When a minimum utility threshold is set to larger than or equal to 0.03%, previous tree-based algorithms, UP-Growth (Tseng et al., 2010) and UP-Growth+(Tseng et al., 2013), and MU-Strategy 2 consume similar runtimes. In addition, MU-Growth and MU-Strategy 1 show the comparable performance with no smaller threshold values than 0.05%. As we can see in Fig. 6 (c), they generate the analogous number of candidates when a minimum utility threshold is no less than 0.05%. It signifies that both methods have similar pruning effects. In like manner, UP-Growth, UP-Growth+, and MU-Strategy 2 prune the comparable number of candidate itemsets when a threshold value is larger than or equal to 0.03%. Overall, our algorithm applying the two pruning strategies is faster than others in discovering the same high utility itemsets by eliminating the more number of candidates as show in Fig 6 (a) and (c). Moreover, we can observe that runtime for mining high utility itemsets is proportional to the number of generated candidate itemsets through the figures. Meanwhile, Fig. 6 (b) and (d) are the experimental results of the total runtime and the number of candidates on Foodmart. In contrast to the experiments on Chain-store, all of the compared algorithms generally operate min-

Table 7
Characteristics of synthetic datasets.

Dataset	|D|	T_{avg}	|I|
T10I4D100K	100,000	10	1000
T10I4D200K	200,000	10	1000
T10I4D400K	400,000	10	1000
T10I4D600K	600,000	10	1000
T10I4D800K	800,000	10	1000
T10I4D1000K	1,000,000	10	1000
T10N10000L1000	100,000	10	10,000
T20N20000L2000	100,000	20	20,000
T30N30000L3000	100,000	30	30,000
T40N40000L4000	100,000	40	40,000

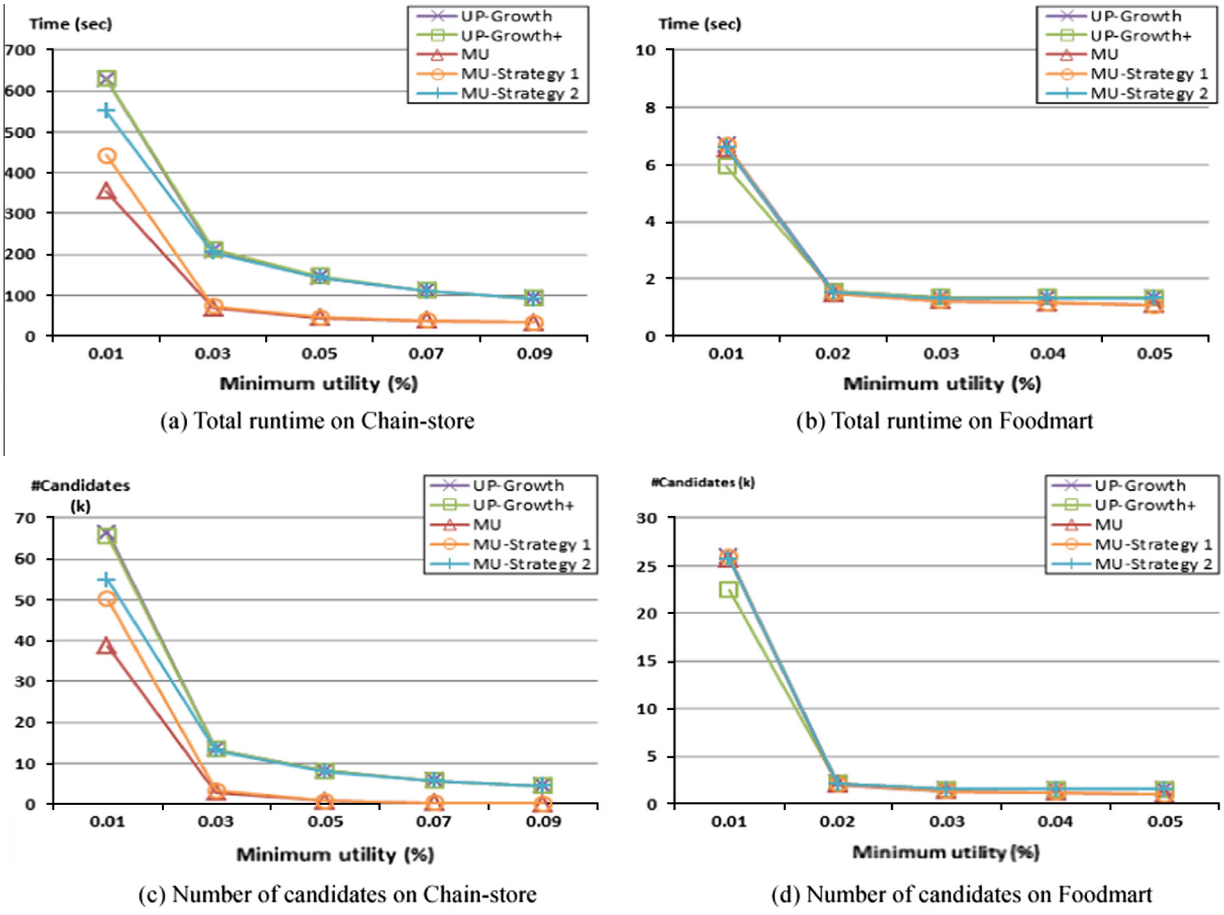


Fig. 6. Performance comparison on real utility datasets.

ing process in a short time as we can see from Fig. 6 (b). Then, they extract the similar number of candidate itemsets as shown in Fig. 6 (d). The reason is that Foodmart is a small sparse dataset. In addition, the average length of transactions is short and the number of items is small in the dataset. That is, the algorithms not only extract the tiny number of candidates but also identify actual high utility itemsets from the candidates in a short time on the Foodmart dataset when a minimum utility threshold is larger than 0.01%.

Fig. 7 is the experimental results of performance evaluation on real and synthetic sparse datasets, Retail and T10.I4.D100K. Fig. 7 (a) and (c) present the results of the total runtime and the number of generated candidate itemsets on Retail. In Fig. 7 (a), the proposed method, MU-Growth has the best performance in terms of the total runtime. In addition, MU-Strategy 2 performs mining process faster than the others, except for MU-Growth applying the two pruning strategies. MU-Strategy 1 is faster than UP-Growth and UP-Growth+ when a minimum utility threshold is no less than 0.05% and slower than UP-Growth+ when the threshold is set to less than 0.04%. However, MU-Strategy 1 still outperforms UP-Growth with respect to all of the threshold values. From the results on the Retail dataset, we can learn that the total runtime of each method is proportionate to its number of generated candidates. It signifies that reducing the number of candidate itemset is essential in the framework of tree-based high utility pattern mining. MU-Growth prunes candidates most effectively, and as a result it discovers high utility itemsets with the fastest speed. In Fig. 7 (a) and (c), when a minimum utility threshold is set to 0.01%, UP-Growth and UP-Growth+ extract about 3.7 and 2.1 times number of candidate itemsets than MU-Growth. Consequently, they require more runtime (about 3.0 and 1.9 times). In the experimental result in terms of the total runtime on T10.I4.D100K in Fig. 7 (d), the compared methods generate the similar number of candidates in every case. Accordingly, they consume runtime for mining high utility patterns proportionally to the number as shown in Fig. 7 (b).

Next, we present the experimental results of the compared algorithms on real dense datasets, Connect and Accidents, in Fig. 8. Fig. 8 (a) and (b) are the results in terms of consumed runtime for Phase I on Connect and Accidents. In this part, we only show the runtime for Phase I where candidate itemsets are extracted from a database in contrast to the experiments on sparse datasets in Figs. 6 and 7. The reason is that the algorithms generate much more candidates on the dense datasets, Connect and Accidents, than the sparse datasets, Foodmart and T10.I4.D100K, as we can see from the results in Fig. 8 (c) and (d). Moreover, the number and average length of transactions are not small as much as the identifying process is performed fast enough in the dense datasets. Nevertheless, we can judge the results in terms of the total runtime from ones with respect to the number of candidates in Fig. 8 (c) and (d) as we can observe from the previous experiments on the sparse datasets in Figs. 6 and 7. The reason is that the most time-consuming part of mining process is Phase II for identifying actual high utility itemsets from extracted candidates in tree-based high utility itemset mining. First, we compare the experimental results of the compared algorithms on Connect in Fig. 8 (a) and (c) with regard to the total runtime and the number of candidate itemsets. In the results, MU-Growth and MU-Strategy 2 show the almost same performance. That is, with the similar pruning effect, they generate candidate itemsets faster than the other algorithms. Overall performance of MU-Growth and MU-Strategy 2 is the best, followed by MU-Strategy 1 and UP-Growth, and UP-Growth is the

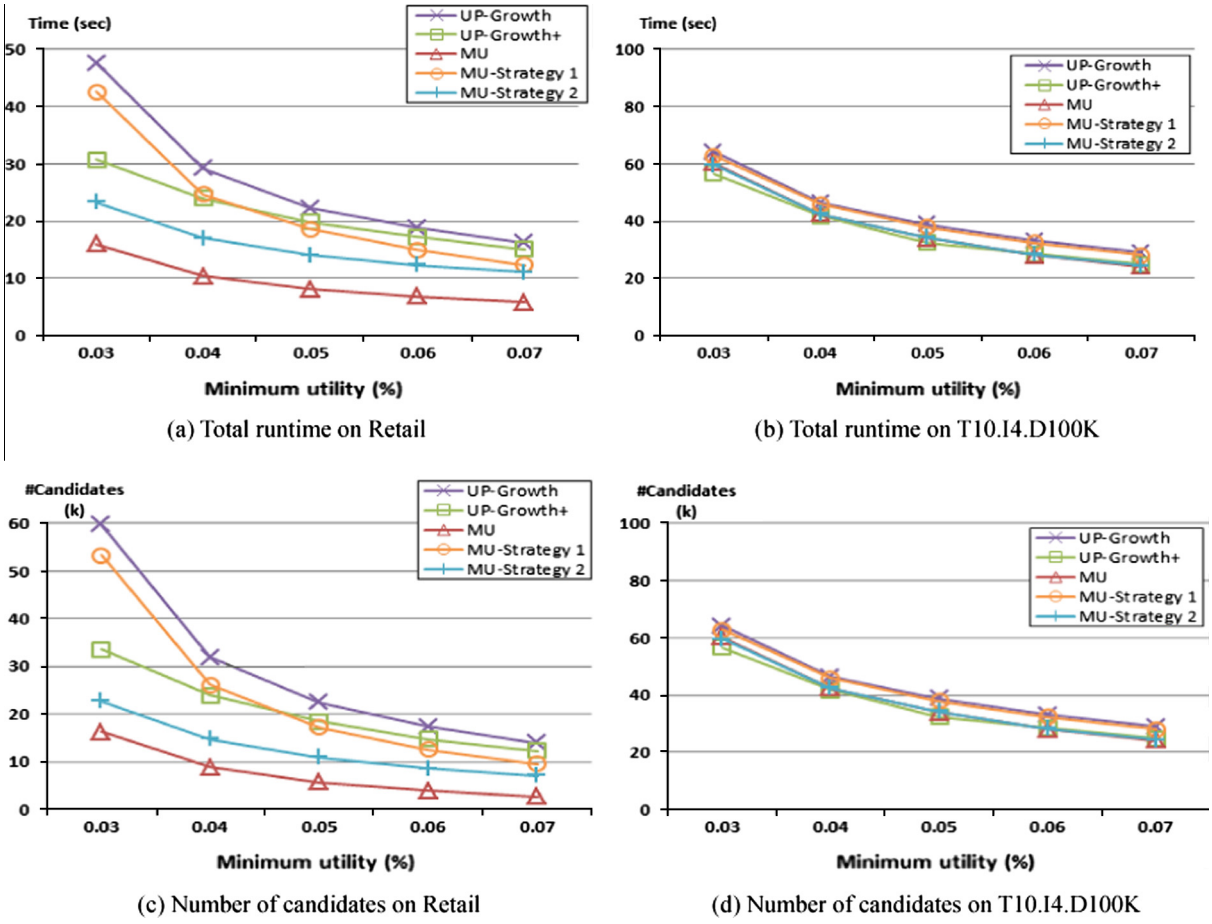


Fig. 7. Performance comparison on real and synthetic sparse datasets.

worst. Meanwhile, Fig. 8 (b) and (d) are the results of performance evaluation on the dense dataset, Accidents. From the figures, we can see that MU-Growth and MU-Strategy 2 show the best performance. It signifies that the second strategy, MU-Strategy 2 have the better effect than the first one for pruning candidates on the Accidents dataset. Overall, MU-Growth and MU-Strategy 2 outperform the others on the dense datasets. Through the experimental results from Figs. 6–8 on the real datasets including the synthetic dataset, T10.I4.D100K, the proposed method, MU-Growth applying the two pruning strategies conducts high utility itemset mining with the fastest speed by eliminating candidate itemsets most effectively. In addition, each pruning strategy, MU-Strategy 1 and 2 show the better performance than UP-Growth in the almost every case. Furthermore, the second strategy, MU-Strategy 2 outperforms UP-Growth+, except for the experiments on the small sparse datasets, Foodmart and T10.I4.D100K. On the dense datasets, it prunes the more number of candidates than the first strategy, MU-Strategy 1.

4.2.2. Performance comparison under varied parameter

We present the experimental results of the compared algorithms under varied maximum number of purchased items in Fig. 9. For this performance evaluation, we use the Accidents dataset and set a minimum utility threshold to 10%. In Fig. 9 (a), the runtime for Phase I of MU-Growth and MU-Strategy 2 is the best, followed by MU-Strategy 1, UP-Growth, and UP-Growth+ is the worst. In the result, the algorithms require more runtime with increasing Max Q. The reason is that minimum and maximum item utilities that are used for reducing overestimated utilities are increased as the maximum quantity value becomes higher. As a result, the decreasing rate of the overestimated utilities drops, which leads to more number of candidates as shown in Fig. 9 (b). In addition, we can observe that MU-Growth and MU-Strategy 2 generate the least number of candidate itemsets. That is, they prune candidates more effectively than the others in mining process.

4.2.3. Scalability of the proposed method

Finally, we evaluate scalability of the algorithms using the synthetic datasets in Table 7. Fig. 10 (a) and (c) are the results in terms of the total runtime and the number of extracted candidates under increasing database size on T10.I4.DxK. In this experiment, a minimum utility threshold is assigned to 0.1%. We can see that all of the methods have a good scalability with respect to raising database size through Fig. 10 (a). Besides, the more runtime is consumed when the total number of transactions is increased in the dataset. The reason for this result is that the algorithms construct global trees by reading transactions and calculate utilities of candidates for identifying actual high utility itemsets. It means that although each method, except for UP-Growth+, generates the similar number of candidate itemsets regardless of the database size as shown in Fig. 10 (c), they need to scan the database for computing actual utilities. Meanwhile, Fig. 10 (b) and (d) show the experimental results under increasing item size on Ta.Nb.Lc. In this experiment, a minimum utility threshold is set to 0.1%. Fig. 10 (b) is the result in terms of the total runtime of the compared algorithms on the datasets. In the figure, the runtime of MU-Growth is the best, followed by MU-Strategy 2, UP-Growth+, MU-Strategy 1,

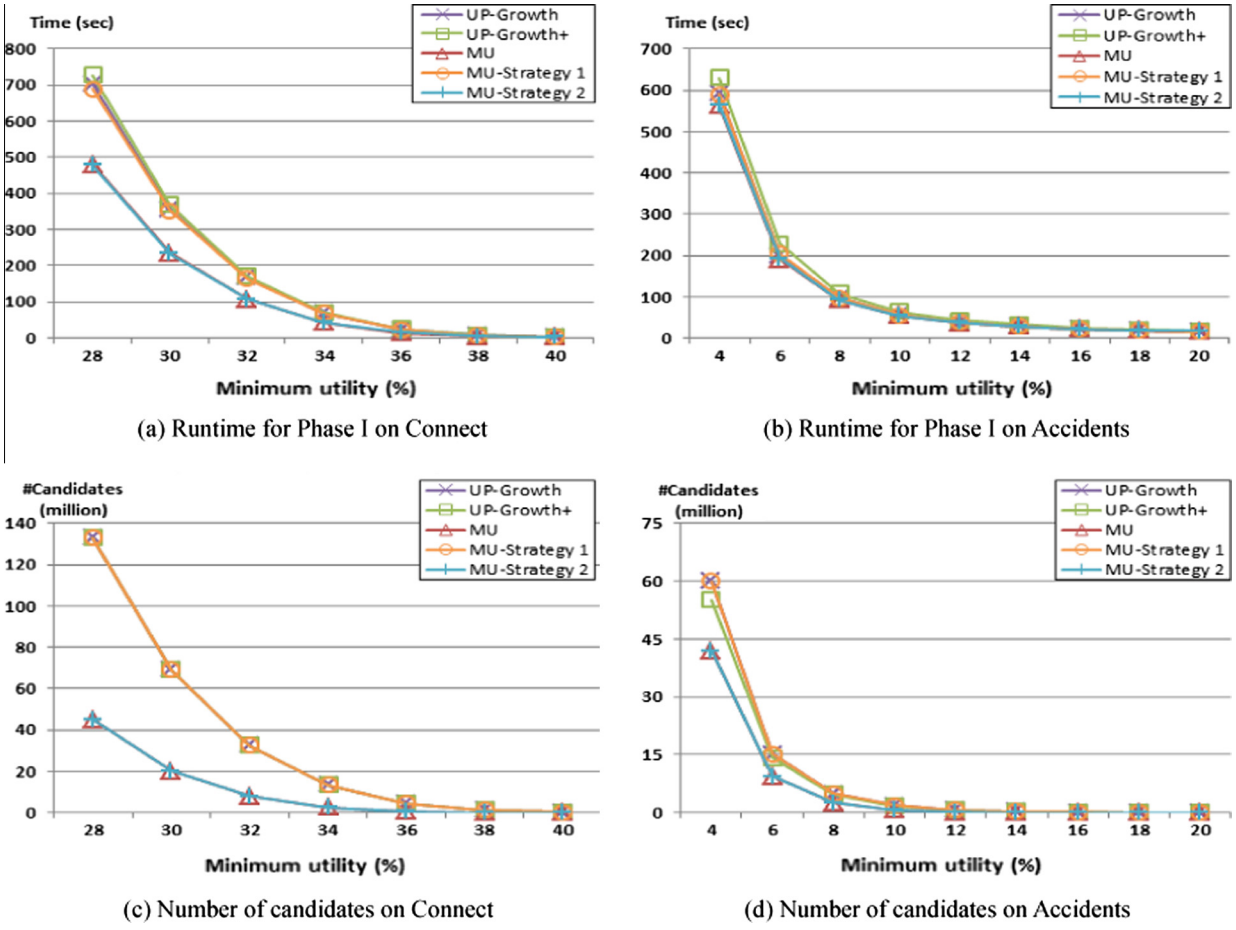


Fig. 8. Performance Comparison on Real Dense Datasets.

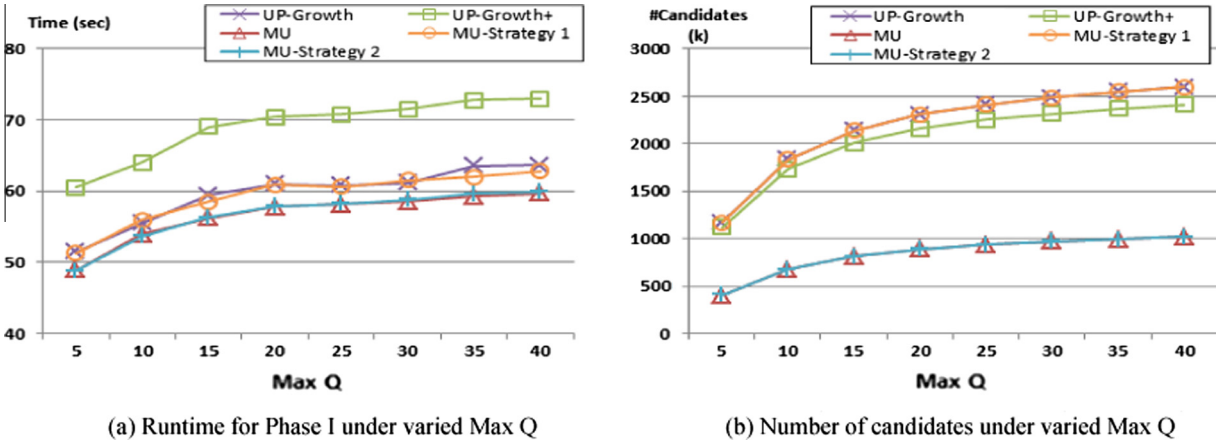


Fig. 9. Performance comparison under varied maximum number of purchased items (accidents).

and UP-Growth is the worst. The reason is that our MU-Growth extracts the least number of candidates as we can see from Fig. 10 (d). That is, it eliminates candidate itemsets most effectively in mining process among the tree-based high utility itemset mining algorithms.

4.2.4. Memory usage of the proposed method

In this part, we present and analyze the memory consumption of the compared methods. Table 8 shows the results of the memory usage on real sparse dataset, Chain-store. In the table, all the algorithms require more memory resources with decreasing minimum utility threshold value. The reason is that the number of promising items that are not pruned is increased in the construction of a global tree when a minimum utility threshold becomes lower. Accordingly, the more number of nodes is created in the tree. Thus, necessary memory space to keep the node information is raised. Furthermore, UP-Growth+ uses more memory resources than other algorithms, UP-Growth and MU-Growth since it store minimal node utilities to nodes. Meanwhile, Table 9 is the memory usage result of the compared methods on real dense dataset, Connect. In the result, memory consumption is increased with decreasing a threshold value like as the experiment on Chain-store. In contrast

Table 8
Memory usage under varied minimum utility on chain-store.

Minutil (%)	UP-Growth	UP-Growth+	MU
0.09	465.004	494.527	465.230
0.07	508.730	541.320	508.996
0.05	561.066	597.375	561.379
0.03	620.555	661.156	620.832
0.01	688.359	733.750	688.633

Table 9
Memory usage under varied minimum utility on connect.

Minutil (%)	UP-Growth	UP-Growth+	MU
40	7.434	7.258	7.426
38	7.430	7.555	7.453
36	8.559	8.754	8.555
34	8.555	8.805	8.555
32	9.633	9.953	9.652
30	10.258	10.605	10.297
28	13.832	14.398	13.803

to the Chain-store dataset, there is less memory requirement because many nodes are shared due to transactions having a similar form in the Connect dataset. From the overall results, we can learn that MU-Growth performs high utility itemset mining faster than UP-Growth and UP-Growth+ with similar or less memory usage.

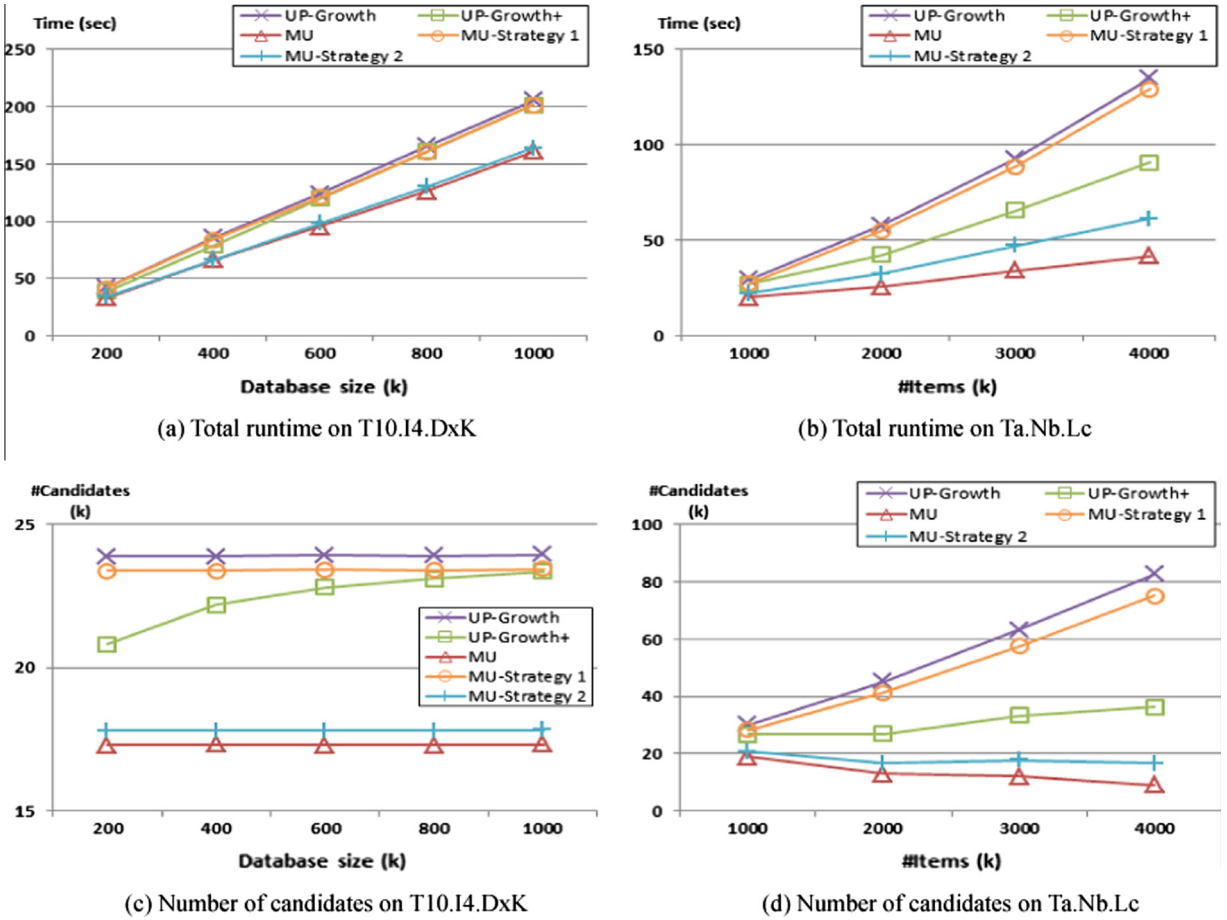
4.3. Performance comparison with IHUP

Settings for the performance evaluation with IHUP (Ahmed et al., 2009) are as follows. Initially, MIQ-Tree and IHUP-Tree are constructed with a single database scan. Next, items in the transactions are rearranged according to TWU descending order during the process of restructuring the trees. Third, candidates are extracted from the restructured trees at Phase I. Finally, the algorithms identify actual high utility itemsets by scanning the original databases at Phase II.

4.3.1. Performance comparison on different datasets

We first compare runtime and the number of candidates with real sparse datasets, Chain-store and Foodmart. As shown in Fig. 11 (e), more than two times the number of candidate itemsets is generated by IHUP when the minimum utility threshold is between 0.01% and 0.03%. The reason is that MU-Growth effectively prunes candidates in the mining process. Fig. 11 (a) and (c) shows runtimes on Chain-store for Phase I and II respectively. In the figures, the overall performance of the proposed method outperforms IHUP. It is because the more number of candidates is extracted by IHUP, and thus the more runtime is consumed. Especially when the minimum threshold is 0.01%, IHUP requires about 2.3 times more runtime than MU-Growth since it generates about 2.3 times more the number of candidates on the threshold. Meanwhile, Fig. 11 (b), (d), and (f) show experimental results on Foodmart dataset. In the results in terms of runtime, MU-Growth outperforms IHUP for Phase I and II. In Fig. 11 (d), the algorithms have almost the same performance when *minutil* is larger than 0.02% due to the small number of generated candidates, and the reason is that the size of Foodmart dataset is small.

We show the performance of the compared algorithms on the synthetic dataset under varied minimum utility threshold, and Fig. 12 is the result on T10.I4.D100K. In Fig. 12 (c), MU-Growth



(a) Total runtime on T10.I4.DxK

(b) Total runtime on Ta.Nb.Lc

(c) Number of candidates on T10.I4.DxK

(d) Number of candidates on Ta.Nb.Lc

Fig. 10. Experimental results of scalability evaluation.

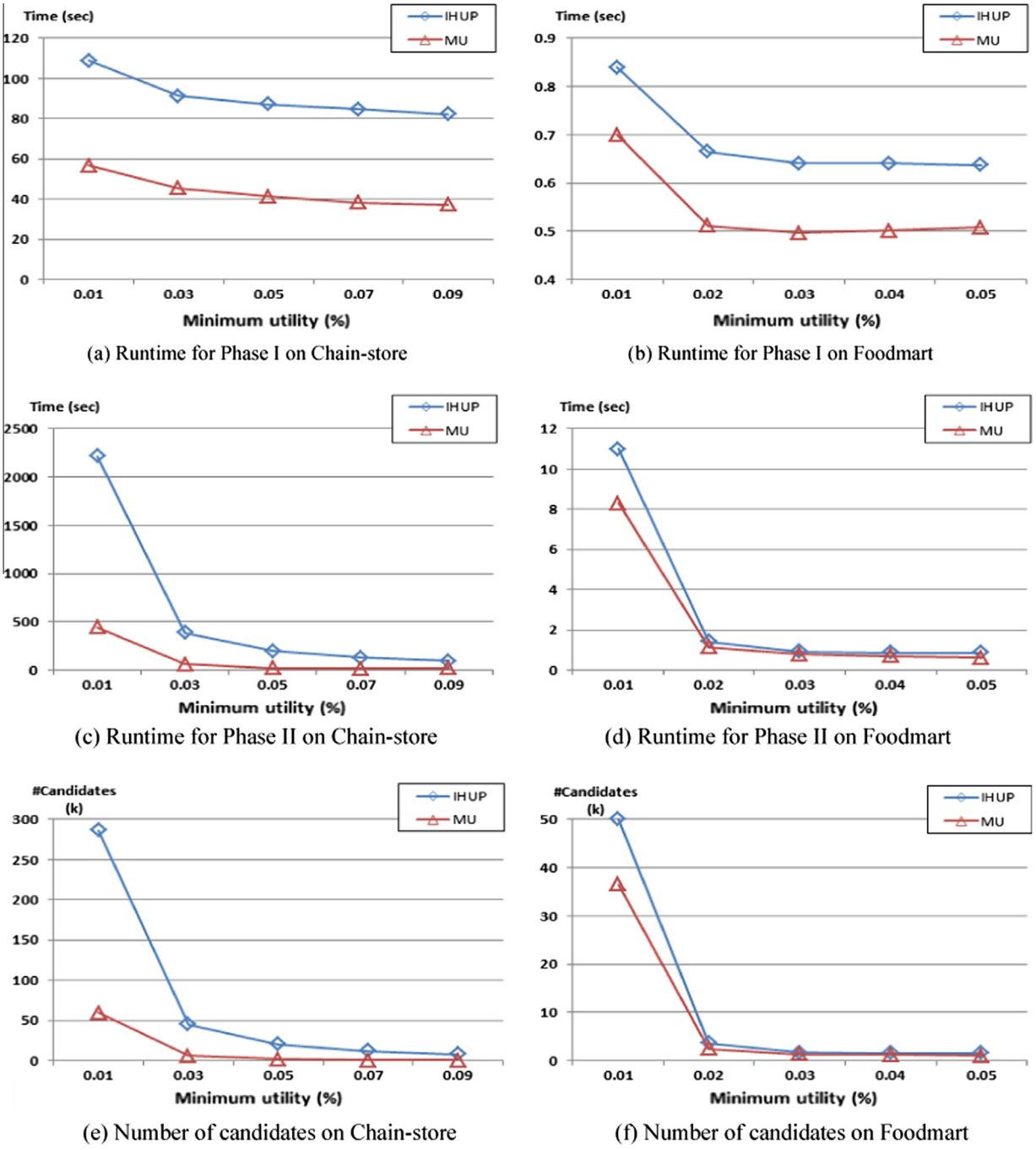


Fig. 11. Performance comparison on sparse utility datasets.

generates fewer candidates than IHUP. The reason is that the proposed two pruning strategies effectively reduce candidates in mining process. Moreover, IHUP uses overestimated utility model and generates almost all possible candidates. On the other hand, Fig. 12 (a) and (b), shows the runtime for Phase I and II. In the result, runtimes of the proposed method outperform those of IHUP. We can see that the runtime for Phase II is longer than for Phase I. In addition, the more number of candidates consumes the more execution time in Phase II. From the above results, we can learn that most total runtimes highly depend on Phase II. The reason is that the overhead of scanning database is huge. It means that reducing the number of candidates is essential issue in utility mining.

The experimental results on real dense datasets, Connect and Accidents, are shown in Fig. 13. In the experiment, we show runtime for Phase I and the number of candidates except runtime for Phase II. The reason is that the runtime for Phase II is very long when database is large and dense. However, we can assume the runtime through the results on the number of candidates. It is because runtime for Phase II is proportional to the number of candidates. In Fig. 13 (a), runtime of MU-Growth for Phase I outperforms that of IHUP. The reason is that it requires more execution time since it generates almost all possible candidates on Phase I. Especially, our method substantially outperforms in terms of the number of candidates as shown in Fig. 13 (c). As a result, we can assume that the huge amount of runtime is consumed by IHUP. We can

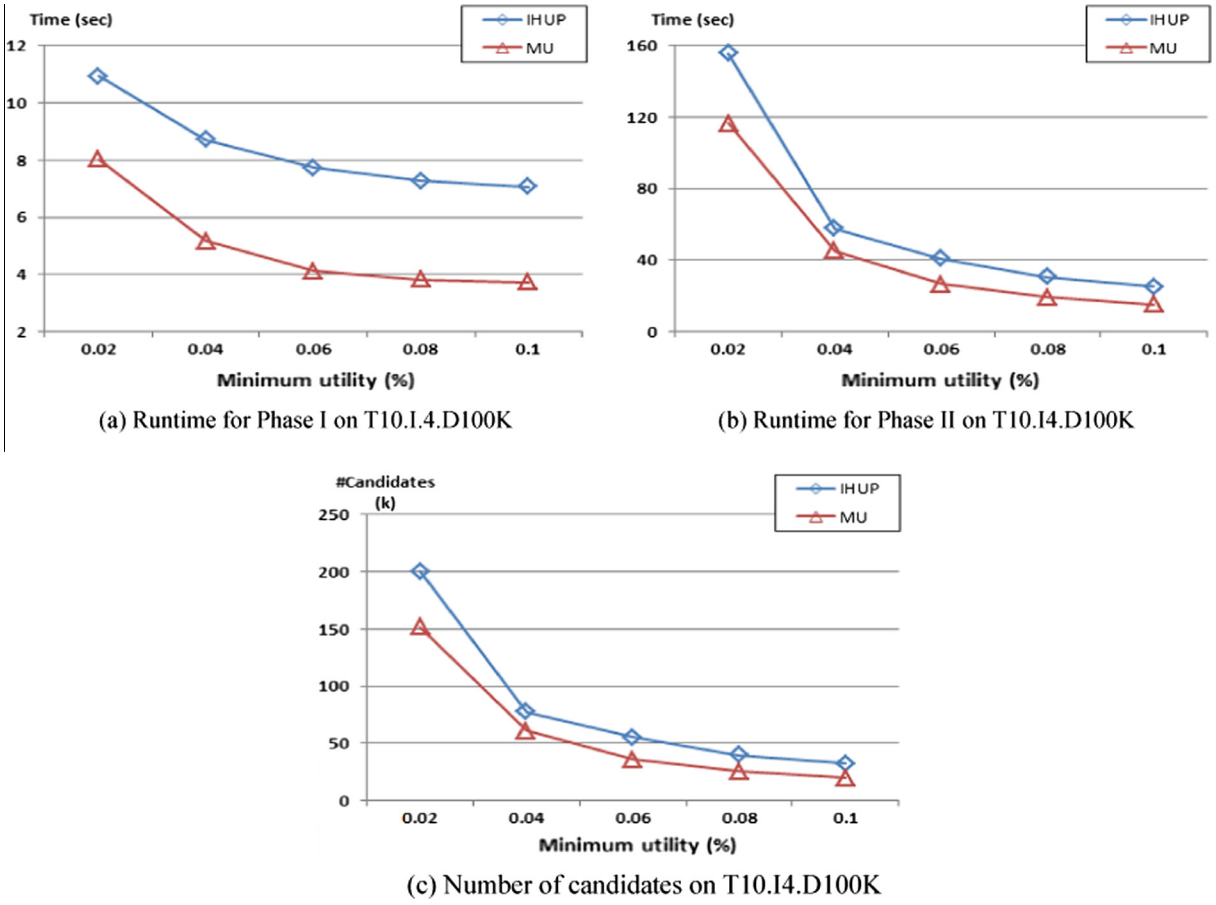


Fig. 12. Performance comparison on synthetic dataset.

learn that MU-Growth effectively reduces the number of candidates and makes the performance much better than the IHUP algorithm through all of the experimental results. Meanwhile, Fig. 13 (b) and (d) show the results on Accident dataset. In Fig. 13 (d), the number of generated candidates of our proposal is significantly smaller than that of IHUP. In addition, the overall performance of IHUP is the worst because it generates most candidates. Through all of the experimental results, we can observe that MU-Growth effectively reduces the number of candidates and make the performance much better than the IHUP algorithm.

4.3.2. Performance comparison under varied strategies

MU-Growth employs two strategies for effectively pruning candidate itemsets. In this part, we use Retail dataset to compare the performance of the strategies. Fig. 14 shows experimental results of IHUP, MU-Growth, and each strategy of MU-Growth. The first strategy is pruning 1-itemset candidates with real item utility and the second one is pruning candidates using estimated maximum itemset utility. We use notations MU-Strategy 1 and 2 for each strategy. Fig. 14 (a) and (b) show the runtimes for Phase I and II, respectively. We can observe that the second strategy outperforms the first one in terms of runtime. In addition, all of the strategies show the similar runtime for Phase I. The reason is that the more runtime is consumed for identifying high utility itemsets than generating candidate itemsets, and the strategies are more effective on pruning process. MU-Growth with the two strategies and MIQ-Tree shows the best performance, followed by strategy 2, 1, and the IHUP is the worst. In Fig. 14 (c), the strategy 2 generates much fewer candidates than the first one. Moreover, MU-Growth shows the best performance in Fig. 14 since it applies all of the strategies. Although there are differences of the performance between the strategies, they still outperform IHUP in all experiments because techniques for reducing overestimated utilities are applied to the proposed tree structures and each strategy effectively decreases the number of candidate itemsets.

4.3.3. Performance comparison under varied parameter

In this part, we compare the performance under varied parameter. Fig. 15 shows the result under varied maximum number of purchased items on Accidents dataset. In the experiment, *minutil* is set to 10%. In the figure, runtime is proportional to the number of candidate itemsets. The runtime of MU-Growth is the best in the experiment. We can see that runtime of the proposed method is raised with increasing maximum item quantity, Max Q. The reason is that MIQ-Tree is constructed with item quantities and node utilities of the tree are calculated using the information. Moreover, MU-Growth computes estimated utilities with minimum and maximum item utilities, and the utilities can be obtained by the product of external and internal utilities, which are stored as node quantities. That is, with increasing item quantities, not only the node utilities but also the estimated utilities of candidates are also raised, and thus the number of pruned candidates is reduced. On the other hand, runtime of IHUP is almost the same regardless of the parameter since item quantity is not a factor of computation in IHUP. Nevertheless, upper bound of the proposed method is still under IHUP.

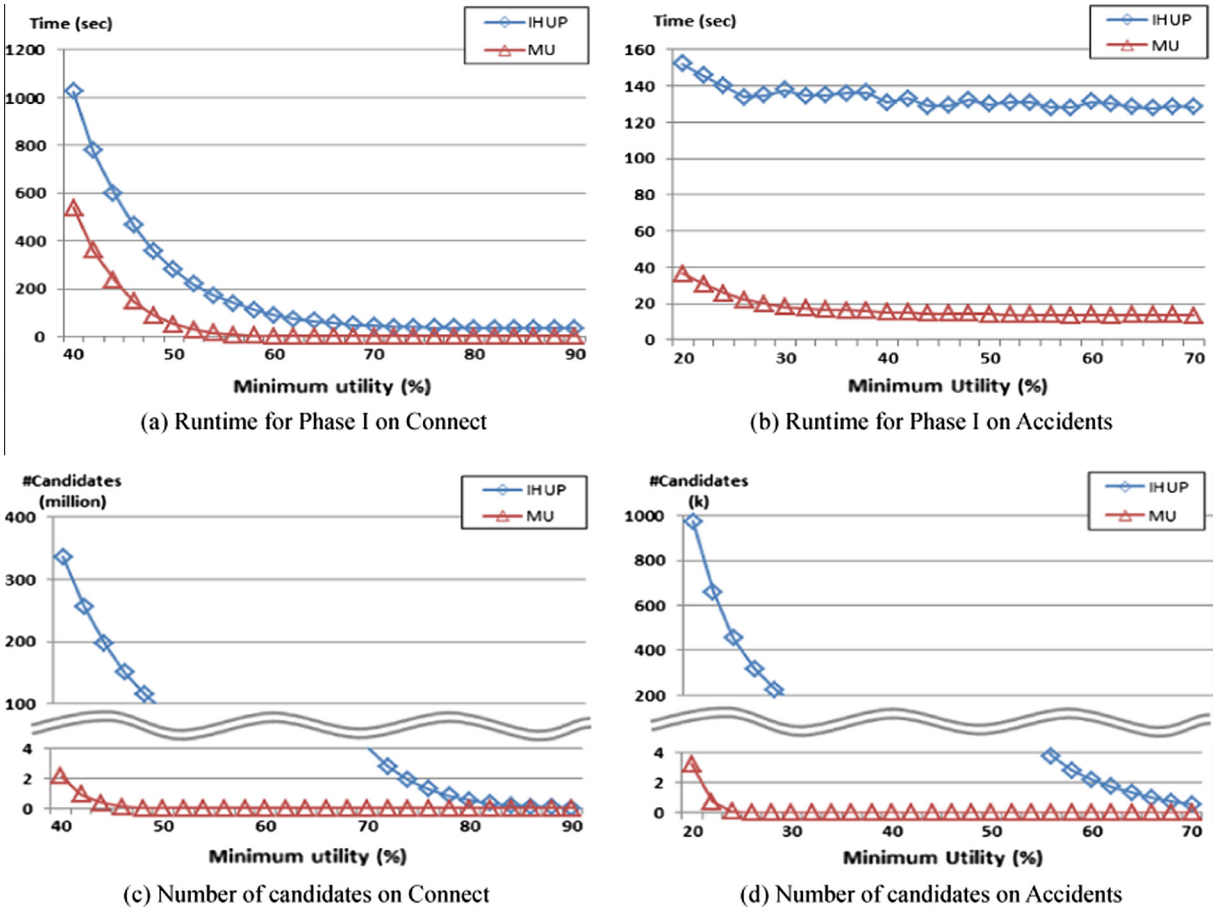


Fig. 13. Performance comparison on dense datasets.

4.3.4. Scalability of the proposed method

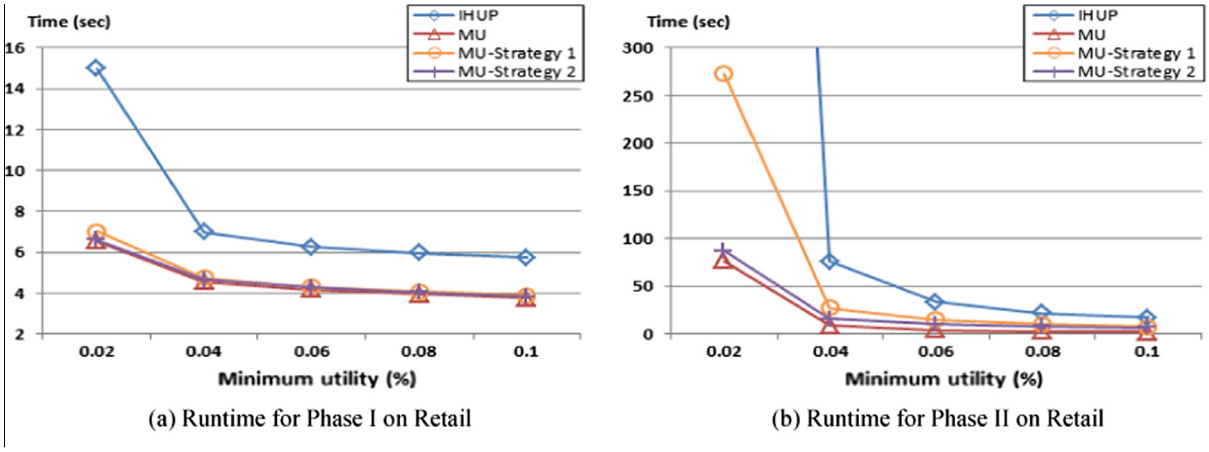
In this subsection, we conduct scalability evaluation for the compared algorithms, MU-Growth and IHUP, using synthetic datasets in Table 7. The experiments are performed in terms of the total runtime and the number of candidates, and *minutil* is set to 0.1%. Fig. 16 (a) and Table 10 show the results under varied database size on T10.I4.DxK. In the figure, we can observe that all of the algorithms have good scalability on runtime. In addition, the total runtime of the proposed method is better than that of IHUP, and it increases when database size is raised. This is because when the size of database increases, runtime for generating candidates and identifying actual high utility itemsets are also raised. In Table 10, the more number of candidates are extracted by IHUP than MU-Growth. The overall performance of the proposed method is the best with increasing size of databases. The reason is that MU-Growth with MIQ-Tree generates the least number of candidates in Phase I. Meanwhile, Fig. 16 (b) and Table 11 show the results under varied item size on Ta.Nb.Lc. In Fig. 16 (b), the total runtime of our proposal outperforms that of IHUP, and they show good scalability. The number of generated candidates is shown in Table 11. In the result, MU-Growth produces the smaller number of candidates than IHUP. Especially, the number of candidates of the proposed method is reduced with decreasing number of high utility itemsets. In contrast, the number of candidates increases in IHUP. Although the proposed method generates candidates proportionally to the number of high utility itemsets, they consume more runtime with the increasing item size. We can learn that the proposed method requires more computation time when the item size is raised.

4.3.5. Memory usage of the proposed method

In this part, we show and analyze memory consumption. Tables 12 and 13 show memory usages of the compared methods (in MB) under varied *minutil* on real datasets, Chain-store and Connect. To construct tree structures with a single-pass, any item is not pruned in a global tree of the compared methods. Therefore, memory usages are similar under varied *minutil* in contrast to methods with multiple database scans in tree construction. From the results, we can learn that all of the high utility itemsets are generated faster than a previous algorithm with a similar memory usage by using the proposed.

5. Discussion

Many applications in the real world include incremental databases as well as static ones. In this environment, important items having no smaller values than a certain threshold can be useless when new transaction information gradually added, and vice versa. Algorithms that construct global trees through two database scans first calculate the importance of items in a database. Then, they prune meaningless items for the construction process. It signifies that the algorithms need to perform scanning the whole database two times again in order to reflect the new information. Therefore, it is necessary to use a data structure created by a single scan for mining high utility itemsets effectively from the incremental dat-



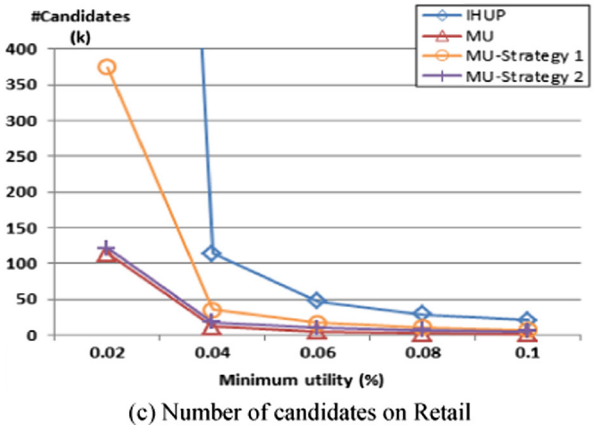


Fig. 14. Experimental result under varied strategies.

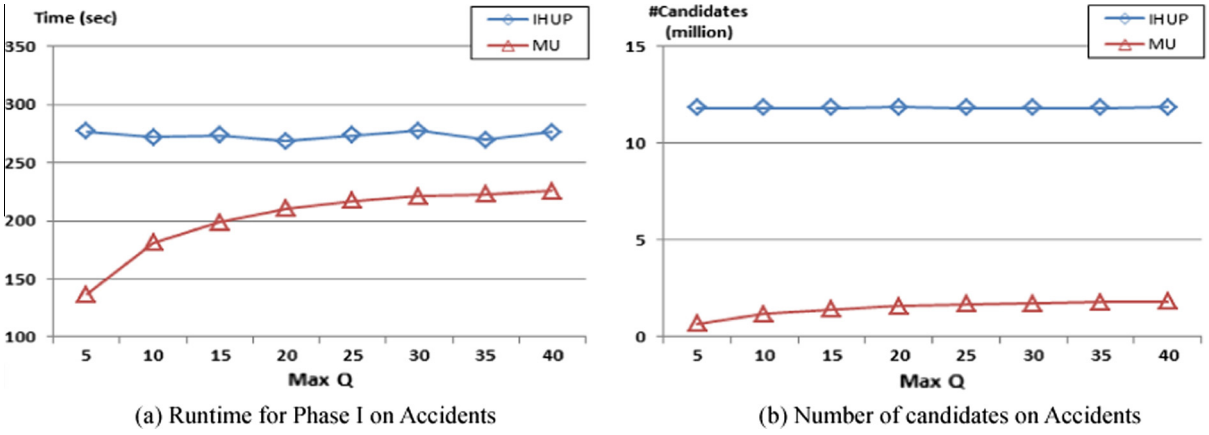


Fig. 15. Varied maximum number of purchased items.

abases. Tree structures with a single-pass construction are generally restructured by arranging nodes in a base sorting order. In here, the sorting order is used as a criterion for efficient itemset mining. Hence, the restructuring methods can be applied to reorder nodes according to the effective sorting order after adding new information to the tree structures. IHUP-Tree (Ahmed et al., 2009) is created with a single database scan and restructured according to a TWU descending order, and thus it is suitable for incremental mining. However, it generates the enormous number of candidate itemsets by applying the overestimated method (Liu et al., 2005). Although UP-Tree (Tseng et al., 2010; Tseng et al., 2013) can extract the decreased number of candidates by reducing overestimated utilities than the IHUP-Tree, it is difficult to adopt the strategies while rearranging nodes according to the order with a single database scan. The proposed MIQ-Tree can not only employ the strategies used in UP-Growth (Tseng et al., 2010) and UP-Growth+ (Tseng et al., 2013) for decreasing overestimated utilities but also rearrange nodes according to a TWU descending order by restructuring method. Therefore, we can apply this study based on the MIQ-Tree to the incremental mining as a future work. Moreover, the second pruning strategy of our approach calculates maximum utility of each expanded itemset from the current prefix with supports in mining process. Through the strategy, we can effectively eliminate search space in the process. A list-based algo-

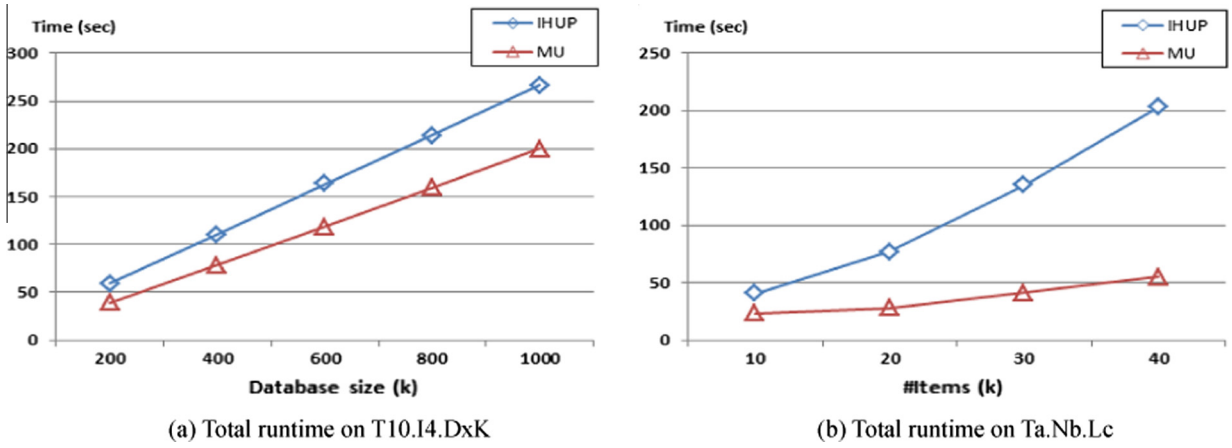


Fig. 16. Experimental results of scalability evaluation.

Table 10
Number of candidates and high utility itemsets under varied database size.

Database	IHUP	MU	#HUIs
200K	32,980	21,165	5123
400K	32,842	22,562	5105
600K	32,838	23,168	5112
800K	32,839	23,562	5106
1000K	32,840	23,792	5106

Table 11
Number of candidates and high utility itemsets under varied item Size.

Items	IHUP	MU	#HUIs
10K	41,118	22,483	10,438
20K	51,581	12,927	6090
30K	73,178	12,183	4812
40K	92,372	9170	3364

Table 12
Memory Usage under varied Minimum Utility on Chain-store.

Minutil (%)	IHUP	MU
0.09	833.914	832.848
0.07	833.816	832.793
0.05	833.824	832.820
0.03	833.777	832.793
0.01	833.742	832.801

Table 13
Memory usage under varied minimum utility on connect.

Minutil (%)	IHUP	MU
90	139.031	138.574
80	139.035	138.602
70	139.008	138.621
60	139.004	138.609
50	139.078	138.633
40	139.063	138.645

rithm, HUI-Miner (Liu M., 2012) has been proposed for mining high utility itemsets. It estimates utility of each extended itemset from the current prefix utilizing utility information of remaining itemset that can be included into the itemset, through which the algorithm prunes search space. Therefore, the algorithm can be improved by applying a technique of estimating maximum utility of each expanded itemset from the current prefix with supports used by the second proposed pruning strategy. Meanwhile, the experimental results in Section 4 showed that MU-Growth with the proposed pruning strategies discovers high utility itemsets faster than state-of-the-art tree-based algorithms (Ahmed et al., 2009; Tseng et al., 2010; Tseng et al., 2013) by eliminating candidate itemsets effectively. By applying the proposed strategies to tree-based algorithms with overestimated methods such as UP-Growth+, we can improve the mining performance. Moreover, if algorithms store candidate itemsets in memory like the experiment in the literature (Liu & Qu, 2012), MU-Growth can mine high utility itemsets more efficiently with less memory usage than other tree-based algorithms due to its pruning effect.

6. Conclusion

In this paper, we proposed a tree-based algorithm with overestimated method, called MU-Growth, for mining high utility itemsets and suggested two strategies for pruning candidates effectively in mining process. By adopting the pruning techniques, we can decrease the number of candidate itemsets and enhance the performance of utility mining frameworks that use tree-based data structures with overestimated methods. Moreover, we also proposed a tree structure, named MIQ-Tree, which is constructed with a single-pass. This tree structure can be restructured without an additional database scan, and the restructuring method employs the concept of reducing overestimated utilities. Therefore, we can facilitate applying the more decreased overestimated utilities than TWU ones to the frameworks with a single-pass tree construction. The experimental results showed that the proposed method improved the performance by reducing the number of candidate itemsets. In addition, MU-Growth outperformed tree-based state-of-the-art algorithms with overestimated methods substantially when databases contain a large number of long transactions or a minimum utility threshold becomes lower.

Acknowledgements

This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF Nos. 2013005682 and 20080062611).

References

Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. In Proc. of the 20th International Conf. on Very Large Data Bases (VLDB 1994), pp. 487–499.

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Choi, H.-J. (2012). Interactive mining of high utility patterns over data streams. *Expert Systems with Applications, 39*(15), 11979–11991.

Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering, 21*(12), 1708–1721.

Barber, B., & Hamilton, H. J. (2003). Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery, 7*(2), 153–185.

Caldersa, T., Dextersb, N., Gillisc, J. J. M., & Goethalsb, B. (2014). Mining frequent itemsets in a stream. *Information Systems, 39*, 233–255.

Chang, L., Wang, T., Yang, D., Luan, H., & Tang, S. (2009). Efficient algorithms for incremental maintenance of closed sequential patterns in large databases. *Data and Knowledge Engineering, 68*(1), 68–106.

Chuang, K.-T., Huang, J.-L., & Chen, M.-S. (2008). Mining top-k frequent patterns in the presence of the memory constraint. *The International Journal on Very Large Data Bases, 17*(5), 1321–1344.

Duonga, H., Truonga, T., & Vob, B. (2014). An efficient method for mining frequent itemsets with double constraints. *Engineering Applications of Artificial Intelligence, 27*, 148–154.

Erwin, A., Gopalan, R. P., & Achuthan, N. R. (2008). Efficient mining high utility itemsets from large datasets. *Advances in Knowledge Discovery and Data Mining (PAKDD 2008)*, 554–561.

Han, J., Pei, J., & Yin Y. (2000). Mining Frequent Patterns without Candidate Generation. In Proc. of the 2000 ACM SIGMOD International Conf. on Management of Data, pp. 1–12.

Hong, T.-P., Lee, C.-H., & Wang, S.-L. (2011). Effective utility mining with the measure of average utility. *Expert Systems with Applications, 38*(7), 8259–8265.

Lee, D., Park, S.-H., & Moon, S. (2013). Utility-based association rule mining: A marketing solution for cross-selling. *Expert Systems with Applications, 40*(7), 2715–2725.

Lee, G., Yun, U., & Ryu, K. (2014). Sliding window based weighted maximal frequent pattern mining over data streams. *Expert Systems with Applications, 41*(2), 694–708.

Leung, C. K.-S., Khan, Q. I., Li, Z., & Hoque, T. (2007). CanTree: A canonical-order tree for incremental frequent-pattern mining. *Knowledge and Information Systems, 11*(3), 287–311.

Li, Y.-C., Yeh, J.-S., & Chang, C.-C. (2008). Isolated items discarding strategy for discovering high utility itemsets. *Data and Knowledge Engineering, 61*(1), 198–217.

Lin, C.-W., Hong, T.-P., & Lu, W.-H. (2011). An effective tree structure for mining high utility itemsets. *Expert Systems with Applications, 38*(6), 7419–7424.

Lin, C.-W., Lan, G.-C., & Hong, T.-P. (2012). An incremental mining algorithm for high utility itemsets. *Expert Systems with Applications, 39*(8), 7173–7180.

Lin, M.-Y., Tu, T.-F., & Hsueh, S.-C. (2012). High utility pattern mining using the maximal itemset property and lexicographic tree structures. *Information Sciences, 215*, 1–14.

Liu, M., & Qu J.-F. (2012). Mining high utility itemsets without candidate generation. International Conference on Information and Knowledge Management (CIKM 2012), pp. 55–64.

Liu, Y., Liao, W.-K., & Choudhary A.N. (2005). A two-phase algorithm for Fast discovery of high utility itemsets. In Advances in Knowledge Discovery and Data Mining (PAKDD 2005), pp. 689–695.

Liu, J., Wang, K., & Fung, B. C. M. (2012). Direct discovery of high utility itemsets without candidate generation. In Proc. of the 2012 IEEE International Conf. on Data Mining (ICDM 2012), pp. 984–989.

Pisharath, J., Liu, Y., Liao, W. K., Choudhary, A., Memik, G., & Parhi. J. (2005). NU-MineBench version 2.0 dataset and technical report, http://cucis.ece.northwestern.edu/projects/DMS /MineBench.html.

Pyun, G., Yun, U., & Ryu, K. (2014). Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems, 55*, 125–139.

Shie, B.-E., Hsiao, H.-F., & Tseng, V. S. (2013). Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments. *Knowledge and Information Systems, 37*(2), 363–387.

Shie, B.-E., Hsiao, H.-F., Tseng, V. S., & Yu, P. S. (2011). Mining high utility mobile sequential patterns in mobile commerce environments. *Database Systems for Advanced Applications (DASFAA 2011)*, 224–238.

Shie, B.-E., Yu, P. S., & Tseng, V. S. (2012). Efficient algorithms for mining maximal high utility itemsets from data streams with different models. *Expert Systems with Applications, 39*(17), 12947–12960.

Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient single-pass frequent pattern mining using a prefix-tree. *Information Sciences, 179*(5), 559–583.

Tseng, V. S., Wu, C. -W., Shie, B. -E., & Yu, P. S. (2010). UP-Growth: An efficient algorithm for high utility itemset mining. In Proc. of the 16th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining (KDD 2010), pp. 253–262.

Tseng, V. S., Shie, B.-E., Wu, C.-W., & Yu, P. S. (2013). Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering, 25*(8), 1772–1786.

Vo, B., Coenen, F., & Le, Bac (2013). A new method for mining frequent weighted itemsets based on wit-trees. *Expert Systems with Applications, 40*(4), 1256–1264.

Wang, E. T., & Chen, A. L. P. (2011). Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis. *Data Mining and Knowledge Discovery, 23*(2), 252–299.

Weiss, G. M., Zadrozny, B., & Saar-Tsechansky, M. (2008). Guest editorial: Special issue on utility-based data mining. *Data Mining and Knowledge Discovery, 17*(2), 129–135.

Wu, C. -W., Fournier-Viger, P., Yu, P. S., & Tseng, V. S. (2011). Efficient Mining of a Concise and Loss-less Representation of High Utility Itemsets. The 11th IEEE International Conf. on Data Mining (ICDM 2011), pp. 824–833.

Wu, C.-W., Lin, Y.-F., Yu, P. S., & Tseng, V. S. (2013). Mining high utility episodes in complex event sequences. *Knowledge Discovery and Data Mining (KDD 2013)*, 536–544.

Wu, C.-W., Shie, B.-E., Tseng, V. S., & Yu, P. S. (2012). Mining top-K high utility itemsets. *Knowledge Discovery and Data Mining (KDD 2012)*, 78–86.

Yeh, J.-S., Li, Y.-C., & Chang, C.-C. (2007). Two-phase algorithms for a novel utility-frequent mining model. *Emerging Technologies in Knowledge Discovery and Data Mining (PAKDD 2007)*, 433–444.

Yin, J., Zheng, Z., & Cao, L. (2012). USpan: An efficient algorithm for mining high utility sequential patterns. *Knowledge Discovery and Data Mining (KDD 2012)*, 660–668.

Yun, U., Lee, G., & Ryu, K. (2014). Mining maximal frequent patterns by considering weight conditions over data streams. *Knowledge-Based Systems, 55*, 49–65.

Yun, U., & Ryu, K. (2013). Efficient mining of maximal correlated weight frequent patterns. *Intelligent Data Analysis, 17*(5).