Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases

Vincent S. Tseng, Bai-En Shie, Cheng-Wei Wu, and Philip S. Yu, *Fellow*, *IEEE*

Abstract—Mining high utility itemsets from a transactional database refers to the discovery of itemsets with high utility like profits. Although a number of relevant algorithms have been proposed in recent years, they incur the problem of producing a large number of candidate itemsets for high utility itemsets. Such a large number of candidate itemsets degrades the mining performance in terms of execution time and space requirement. The situation may become worse when the database contains lots of long transactions or long high utility itemsets. In this paper, we propose two algorithms, namely *utility pattern growth* (*UP-Growth*) and UP-Growth+, for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets. The information of high utility itemsets is maintained in a tree-based data structure named *utility pattern tree* (*UP-Tree*) such that candidate itemsets can be generated efficiently with only two scans of database. The performance of UP-Growth and UP-Growth+ is compared with the state-of-the-art algorithms on many types of both real and synthetic data sets. Experimental results show that the proposed algorithms, especially UP-Growth+, not only reduce the number of candidates effectively but also outperform other algorithms substantially in terms of runtime, especially when databases contain lots of long transactions.

Index Terms—Candidate pruning, frequent itemset, high utility itemset, utility mining, data mining

1 Introduction

Data mining is the process of revealing nontrivial, previously unknown and potentially useful information from large databases. Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, weighted frequent pattern mining, and high utility pattern mining. Among them, frequent pattern mining is a fundamental research topic that has been applied to different kinds of databases, such as transactional databases [1], [14], [21], streaming databases [18], [27], and time series databases [9], [12], and various application domains, such as bioinformatics [8], [11], [20], Web click-stream analysis [7], [35], and mobile environments [15], [36].

Nevertheless, relative importance of each item is not considered in frequent pattern mining. To address this problem, weighted association rule mining was proposed [4], [26], [28], [31], [37], [38], [39]. In this framework, weights of items, such as unit profits of items in transaction databases, are considered. With this concept, even if some items appear infrequently, they might still be found if they have high weights. However, in this framework, the quantities of items are not considered yet. Therefore, it cannot satisfy the requirements of users who are interested in discovering the itemsets with high sales profits, since the profits are composed of unit profits, i.e., weights, and purchased quantities.

In view of this, utility mining emerges as an important topic in data mining field. Mining high utility itemsets from databases refers to finding the itemsets with high profits. Here, the meaning of itemset utility is interestingness, importance, or profitability of an item to users. Utility of items in a transaction database consists of two aspects: 1) the importance of distinct items, which is called *external utility*, and 2) the importance of items in transactions, which is called *internal utility*. Utility of an itemset is defined as the product of its external utility and its internal utility. An itemset is called a *high utility itemset* if its utility is no less than a user-specified minimum utility threshold; otherwise, it is called a *low-utility itemset*. Mining high utility itemsets from databases is an important task has a wide range of applications such as website click stream analysis [16], [25], [29], business promotion in chain hypermarkets, cross-marketing in retail stores [3], [10], [19], [30], [32], [33], online e-commerce management, mobile commerce environment planning [24], and even finding important patterns in biomedical applications [5].

However, mining high utility itemsets from databases is not an easy task since *downward closure property* [1] in frequent itemset mining does not hold. In other words, pruning search space for high utility itemset mining is difficult because a superset of a low-utility itemset may be a high utility itemset. A naïve method to address this problem is to enumerate all itemsets from databases by the principle of exhaustion. Obviously, this method suffers from the problems of a large search space, especially when databases contain lots of long transactions or a low minimum utility threshold is set. Hence, how to effectively prune the search

- *V.S. Tseng, B.-E. Shie, and C.-W. Wu are with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan City, Taiwan 70101, R.O.C. E-mail: tsengsm@mail.ncku.edu.tw, brianshie@gmail.com, silvemoonfox@idb.csie.ncku.edu.tw.*
- *P.S. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, and the Computer Science Department, King Abdulaziz University, Jeddah, Saudi Arabia. E-mail: psyu@cs.uic.edu.*

Manuscript received 31 Jan. 2011; revised 2 Nov. 2011; accepted 28 Feb. 2012; published online 9 Mar. 2012.
Recommended for acceptance by J. Freire.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-01-0045.
Digital Object Identifier no. 10.1109/TKDE.2012.59.

TABLE 1
An Example Database

TID	Transaction	TU
T_1	(A,1) (C,10) (D,1)	17
T_2	(A,2) (C,6) (E,2) (G,5)	27
T_3	(A,2) (B,2) (D,6) (E,2) (F,1)	37
T_4	(B,4) (C,13) (D,3) (E,1)	30
T_5	(B,2) (C,4) (E,1) (G,2)	13
T_6	(A,1) (B,1) (C,1) (D,1) (H,2)	12

space and efficiently capture all high utility itemsets with no miss is a crucial challenge in utility mining.

Existing studies [3], [10], [16], [17], [19], [24], [29], [30] applied overestimated methods to facilitate the performance of utility mining. In these methods, potential high utility itemsets (PHUIs) are found first, and then an additional database scan is performed for identifying their utilities. However, existing methods often generate a huge set of PHUIs and their mining performance is degraded consequently. This situation may become worse when databases contain many long transactions or low thresholds are set. The huge number of PHUIs forms a challenging problem to the mining performance since the more PHUIs the algorithm generates, the higher processing time it consumes.

To address this issue, we propose two novel algorithms as well as a compact data structure for efficiently discovering high utility itemsets from transactional databases. Major contributions of this work are summarized as follows:

1. Two algorithms, named *utility pattern growth* (*UP-Growth*) and UP-Growth+, and a compact tree structure, called *utility pattern tree* (*UP-Tree*), for discovering high utility itemsets and maintaining important information related to utility patterns within databases are proposed. High-utility itemsets can be generated from UP-Tree efficiently with only two scans of original databases.
2. Several strategies are proposed for facilitating the mining processes of UP-Growth and UP-Growth+ by maintaining only essential information in UP-Tree. By these strategies, overestimated utilities of candidates can be well reduced by discarding utilities of the items that cannot be high utility or are not involved in the search space. The proposed strategies can not only decrease the overestimated utilities of PHUIs but also greatly reduce the number of candidates.
3. Different types of both real and synthetic data sets are used in a series of experiments to compare the performance of the proposed algorithms with the state-of-the-art utility mining algorithms. Experimental results show that UP-Growth and UP-Growth+ outperform other algorithms substantially in terms of execution time, especially when databases contain lots of long transactions or low minimum utility thresholds are set.

The rest of this paper is organized as follows: In Section 2, we introduce the background and related work for high utility itemset mining. In Section 3, the proposed data structure and algorithms are described in details.

TABLE 2
Profit Table

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

Experiment results are shown in Section 4 and conclusions are given in Section 5.

2 Background

In this section, we first give some definitions and define the problem of utility mining, and then introduce related work in utility mining.

2.1 Preliminary

Given a finite set of items $I = \{i_1, i_2, \ldots, i_m\}$, each *item* $i_p (1 \le p \le m)$ has a *unit profit* $pr(i_p)$. An *itemset* X is a set of k distinct items $\{i_1, i_2, \ldots, i_k\}$, where $i_j \in I, 1 \le j \le k$. k is the length of X. An itemset with *length* k is called a k-itemset. A *transaction database* $D = \{T_1, T_2, \ldots, T_n\}$ contains a set of transactions, and each *transaction* $T_d (1 \le d \le n)$ has a unique identifier d, called TID. Each item i_p in transaction T_d is associated with a *quantity* $q(i_p, T_d)$, that is, the purchased quantity of i_p in T_d.

Definition 1. *Utility of an item i_p in a transaction T_d is denoted as $u(i_p, T_d)$ and defined as $pr(i_p) \times q(i_p, T_d)$.*

Definition 2. *Utility of an itemset X in T_d is denoted as $u(X, T_d)$ and defined as $\sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d)$.*

Definition 3. *Utility of an itemset X in D is denoted as $u(X)$ and defined as $\sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$.*

Definition 4. *An itemset is called a* high utility itemset *if its utility is no less than a user-specified minimum utility threshold which is denoted as min_util. Otherwise, it is called a* low-utility itemset.

For example, in Tables 1 and 2,

$$u(\{A\}, T_1) = 5 \times 1 = 5;$$
$$u(\{\mathrm{AD}\}, \mathrm{T}_1) = u(\{\mathrm{A}\}, \mathrm{T}_1) + u(\{\mathrm{D}\}, \mathrm{T}_1) = 5 + 2 = 7;$$
$$u(\{\mathrm{AD}\}) = u(\{\mathrm{AD}\}, T_1) + u(\{\mathrm{AD}\}, T_3) + u(\{\mathrm{AD}\}, T_6)$$
$$= 7 + 22 + 7 = 36.$$

If *min_util* is set to 30, $\{\mathrm{AD}\}$ is a high utility itemset.

Problem statement. Given a transaction database D and a user-specified minimum utility threshold *min_util*, the problem of mining high utility itemsets from D is to find the complete set of the itemsets whose utilities are larger than or equal to *min_util*.

After addressing the definitions about utility mining, we introduce the *transaction-weighted downward closure* (TWDC) which is proposed in [19].

Definition 5. Transaction utility *of a transaction T_d is denoted as $\mathrm{TU}(T_d)$ and defined as $u(T_d, T_d)$.*

Definition 6. Transaction-weighted utility *of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is denoted as $TWU(X)$ and defined as $\sum_{X \subseteq T_d \wedge T_d \in D} \mathrm{TU}(T_d)$.*

Definition 7. *An itemset X is called a* high-transaction-weighted utility itemset *(HTWUI) if* $\mathrm{TWU}(X)$ *is no less than min_util.*

Property 1 (Transaction-weighted downward closure.). *For any itemset X, if X is not a HTWUI, any superset of X is a low utility itemset.*

By Property 1, downward closure property can be maintained in utility mining by applying the transaction-weighted utility. For example, $\mathrm{TU}(\mathrm{T}_2) = u(\{\mathrm{ACEG}\}, \mathrm{T}_2) = 27; \mathrm{TWU}(\{\mathrm{G}\}) = \mathrm{TU}(\mathrm{T}_2) + \mathrm{TU}(\mathrm{T}_5) = 27 + 13 = 40$. If min_util is set to 30, {G} is a HTWUI. However, if *min_util* is set to 50, $\{\mathrm{G}\}$ and its supersets are not HTWUIs since $\mathrm{TWU}(\{\mathrm{G}\})$ must be no less than the TWUs of all $\{\mathrm{G}\}$'s supersets.

2.2 Related Work

Extensive studies have been proposed for mining frequent patterns [1], [2], [13], [14], [21], [22], [34], [40]. Among the issues of frequent pattern mining, the most famous are association rule mining [1], [13], [14], [21], [34], [40] and sequential pattern mining [2], [22]. One of the well-known algorithms for mining association rules is Apriori [1], which is the pioneer for efficiently mining association rules from large databases. Pattern growth-based association rule mining algorithms [14], [21] such as FP-Growth [14] were afterward proposed. It is widely recognized that FP-Growth achieves a better performance than Apriori-based algorithms since it finds frequent itemsets without generating any candidate itemset and scans database just twice.

In the framework of frequent itemset mining, the importance of items to users is not considered. Thus, the topic called *weighted association rule mining* was brought to attention [4], [26], [28], [31], [37], [38], [39]. Cai et al. first proposed the concept of weighted items and weighted association rules [4]. However, since the framework of weighted association rules does not have downward closure property, mining performance cannot be improved. To address this problem, Tao et al. proposed the concept of weighted downward closure property [28]. By using transaction weight, weighted support can not only reflect the importance of an itemset but also maintain the downward closure property during the mining process. There are also many studies [6], [26], [37] that have developed different weighting functions for weighted pattern mining.

Although weighted association rule mining considers the importance of items, in some applications, such as transaction databases, items' quantities in transactions are not taken into considerations yet. Thus, the issue of high utility itemset mining is raised and many studies [3], [5], [10], [16], [17], [19], [24], [25], [29], [30], [32], [33] have addressed this problem. Liu et al. proposed an algorithm named Two-Phase [19] which is mainly composed of two mining phases. In phase I, it employs an Apriori-based level-wise method to enumerate HTWUIs. Candidate itemsets with length k are generated from length k-1 HTWUIs, and their TWUs are computed by scanning the database once in each pass. After the above steps, the complete set of HTWUIs is collected in phase I. In phase II, HTWUIs that are high utility itemsets are identified with an additional database scan.

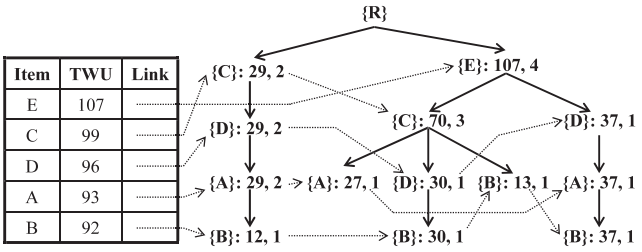


Fig. 1. An IHUP-Tree when $\mathrm{min\_util} = 40$.

Although two-phase algorithm reduces search space by using TWDC property, it still generates too many candidates to obtain HTWUIs and requires multiple database scans. To overcome this problem, Li et al. [17] proposed an isolated items discarding strategy (IIDS) to reduce the number of candidates. By pruning isolated items during level-wise search, the number of candidate itemsets for HTWUIs in phase I can be reduced. However, this algorithm still scans database for several times and uses a candidate generation-and-test scheme to find high utility itemsets.

To efficiently generate HTWUIs in phase I and avoid scanning database too many times, Ahmed et al. [3] proposed a tree-based algorithm, named IHUP. A tree-based structure called IHUP-Tree is used to maintain the information about itemsets and their utilities. Each node of an IHUP-Tree consists of an item name, a TWU value and a support count. IHUP algorithm has three steps: 1) construction of IHUP-Tree, 2) generation of HTWUIs, and 3) identification of high utility itemsets. In step 1, items in transactions are rearranged in a fixed order such as lexicographic order, support descending order or TWU descending order. Then the rearranged transactions are inserted into an IHUP-Tree. Fig. 1 shows the global IHUP-Tree for the database in Table 1, in which items are arranged in the descending order of TWU. For each node in Fig. 1, the first number beside item name is its TWU and the second one is its support count. In step 2, HTWUIs are generated from the IHUP-Tree by applying FP-Growth [14]. Thus, HTWUIs in phase I can be found without generating any candidate for HTWUIs. In step 3, high utility itemsets and their utilities are identified from the set of HTWUIs by scanning the original database once.

Although IHUP achieves a better performance than IIDS and Two-Phase, it still produces too many HTWUIs in phase I. Note that IHUP and Two-Phase produce the same number of HTWUIs in phase I since they both use TWU framework to overestimate itemsets' utilities. However, this framework may produce too many HTWUIs in phase I since the overestimated utility calculated by TWU is too large. Such a large number of HTWUIs will degrade the mining performance in phase I substantially in terms of execution time and memory consumption. Moreover, the number of HTWUIs in phase I also affects the performance of phase II since the more HTWUIs the algorithm generates in phase I, the more execution time for identifying high utility itemsets it requires in phase II.

As stated above, the number of generated HTWUIs is a critical issue for the performance of algorithms. Therefore, this study aims at reducing itemsets' overestimated utilities

and proposes several strategies. By applying the proposed strategies, the number of generated candidates can be highly reduced in phase I and high utility itemsets can be identified more efficiently in phase II.

3 Proposed Methods

The framework of the proposed methods consists of three steps: 1) Scan the database twice to construct a global UP-Tree with the first two strategies (given in Section 3.1); 2) recursively generate PHUIs from global UP-Tree and local UP-Trees by UP-Growth with the third and fourth strategies (given in Section 3.2) or by UP-Growth+ with the last two strategies (given in Section 3.3); and 3) identify actual high utility itemsets from the set of PHUIs (given in Section 3.4). Note that we use a new term "potential high utility itemsets" to distinguish the patterns found by our methods from HTWUIs since our methods are not based on traditional TWU model. By our effective strategies, the set of PHUIs will become much smaller than the set of HTWUIs.

3.1 The Proposed Data Structure: UP-Tree

To facilitate the mining performance and avoid scanning original database repeatedly, we use a compact tree structure, named *UP-Tree*, to maintain the information of transactions and high utility itemsets. Two strategies are applied to minimize the overestimated utilities stored in the nodes of global UP-Tree. In following sections, the elements of UP-Tree are first defined. Next, the two strategies are introduced. Finally, how to construct an UP-Tree with the two strategies is illustrated in detail by a running example.

3.1.1 The Elements in UP-Tree

In an UP-Tree, each node N consists of *N.name*, *N.count*, *N.nu*, *N.parent*, *N.hlink* and a set of child nodes. *N.name* is the node's item name. *N.count* is the node's support count. *N.nu* is the node's *node utility*, i.e., overestimated utility of the node. *N.parent* records the parent node of N. *N.hlink* is a node link which points to a node whose item name is the same as *N.name*.

A table named *header table* is employed to facilitate the traversal of UP-Tree. In header table, each entry records an item name, an overestimated utility, and a link. The link points to the last occurrence of the node which has the same item as the entry in the UP-Tree. By following the links in header table and the nodes in UP-Tree, the nodes having the same name can be traversed efficiently.

In following sections, two strategies for decreasing the overestimated utility of each item during the construction of a global UP-Tree are introduced.

3.1.2 Strategy DGU: Discarding Global Unpromising Items during Constructing a Global UP-Tree

The construction of a global UP-Tree can be performed with two scans of the original database. In the first scan, TU of each transaction is computed. At the same time, TWU of each single item is also accumulated. By TWDC property, an item and its supersets are unpromising to be high utility itemsets if its TWU is less than the minimum utility threshold. Such an item is called an *unpromising item*. Definition 8 gives a formal definition of what are unpromising items and promising items.

Definition 8. *An item i_p is called a* promising item *if* $\mathrm{TWU}(i_p) \geq \mathrm{min\_util}$. *Otherwise it is called an* unpromising item. *Without loss of generality, an item is also called a promising item if its* overestimated utility *(which is different from TWU in this paper) is no less than min_util. Otherwise it is called an unpromising item.*

Property 2 (Antimonotonicity of unpromising items). *If i_u is an unpromising item, i_u and all its supersets are not high utility itemsets.*

Proof. Since i_u is an unpromising item, its overestimated utility, denoted as $\mathrm{OEU}(i_u)$, is less than *min_util*. Also, it is obvious that $u(i_u) \leq \mathrm{OEU}(i_u)$. Thus $u(i_u) < min\_util$, that is, i_u is not a high utility itemset. Assume that the set of supersets of i_u is denoted as SET_u and the set of transactions containing an item x is denoted as T_x. For each itemset i_u' in SET_u, $T_{i_u'} \subseteq T_{i_u}$. Thus, $\mathrm{OEU}(i_u') \leq \mathrm{OEU}(i_u)$. By the above inferences, we get $u(i_u') \leq \mathrm{OEU}(i_u') \leq \mathrm{OEU}(i_u) < min\_util$. Therefore, each superset of i_u is not a high utility itemset. □

Corollary 1. *Only the supersets of promising items are possible to be high utility itemsets.*

During the second scan of database, transactions are inserted into a UP-Tree. When a transaction is retrieved, the unpromising items should be removed from the transaction and their utilities should also be eliminated from the transaction's TU according to Property 2 and Corollary 1. This concept forms our first strategy.

Strategy 1. DGU: Discarding global unpromising items and their actual utilities from transactions and transaction utilities of the database.

Rationale. By Property 2 and Corollary 1, we can realize that unpromising items play no role in high utility itemsets. Thus, when utilities of itemsets are being estimated, utilities of unpromising items can be regarded as irrelevant and be discarded.

New TU after pruning unpromising items is called *reorganized transaction utility* (RTU). RTU of a reorganized transaction T_r is denoted as $\mathrm{RTU}(\mathrm{T}_r)$. By reorganizing the transactions, not only less information is needed to be recorded in UP-Tree, but also smaller overestimated utilities for itemsets are generated. Strategy DGU uses RTU to overestimate the utilities of itemsets instead of TWU. Since the utilities of unpromising items are excluded, RTU must be no larger than TWU. Therefore, the number of PHUIs with DGU must be no more than that of HTWUIs generated with TWU [3], [19]. DGU is quite effective especially when transactions contain lots of unpromising items, such as those in sparse data sets. Besides, DGU can be easily integrated into TWU-based algorithms [3], [19]. Moreover, before constructing an UP-Tree, DGU can be performed repeatedly till all reorganized transactions contain no global unpromising item. By performing DGU for several times, the number of PHUIs will be reduced; however, it needs several database scans.

Subroutine : *Insert_Reorganized_Transaction*(N, i_x)

Line 1 : If N has a child N_{i_x} such that $\mathrm{N}_{i_x}.item = i_x$, increment $\mathrm{N}_{i_x}.count$ by 1. Otherwise, create a new child node N_{i_x} with $\mathrm{N}_{i_x}.item = i_x$, $\mathrm{N}_{i_x}.count = 1$, $\mathrm{N}_{i_x}.parent = \mathrm{N}$ and $\mathrm{N}_{i_x}.nu = 0$.

Line 2 : Increase $\mathrm{N}_{i_x}.nu$ by $(\mathrm{RTU}(t_j') - \sum_{p=x+1}^{n} u(i_p, t_j'))$, where $i_p \in t_j'$.

Line 3 : If $x \neq n$, call *Insert_Reorganized_Transaction*($\mathrm{N}_{i_x}, i_{x+1}$)

Fig. 2. The subroutine of *Insert_Reorganized_Transaction*.

3.1.3 Strategy DGN: Decreasing Global Node Utilities during Constructing a Global UP-Tree

It is shown in [3] that the tree-based framework for high utility itemset mining applies the divide-and-conquer technique in mining processes. Thus, the search space can be divided into smaller subspaces. For example, in Fig. 1, the search space can be divided into the following subspaces:

1. {B}'s conditional tree (abbreviated as {B}-Tree),
2. {A}-Tree without containing {B},
3. {D}-Tree without containing {B} and {A},
4. {C}-Tree without containing {B}, {A}, and {D}, and
5. {E}-Tree without containing {B}, {A}, {D}, and {C}.

It can be observed that in the subspace {A}-Tree, all paths are not related to {B} since the nodes {B} are below the nodes {A} in global IHUP-Tree. In other words, the items that are descendant nodes of the item i_m will not appear in $\{i_m\}$-Tree; only the items that are ancestor nodes of i_m will appear in $\{i_m\}$-Tree. From this viewpoint, our second proposed strategy for decreasing overestimated utilities is to remove the utilities of descendant nodes from their node utilities in global UP-Tree. The process is performed during the construction of the global UP-Tree.

Strategy 2. DGN: Decreasing global node utilities for the nodes of global UP-Tree by actual utilities of descendant nodes during the construction of global UP-Tree.

Rationale. Let $\{i_1, i_2, \ldots, i_n\}$ be an ordered list of promising items. Since items $i_{k+1}, i_{k+2}, \ldots, i_n$ are not involved in i_k-Tree, they will not be contained in any PHUI generated from i_k-Tree. Their utilities can be discarded from node utilities of the nodes about i_k in global UP-Tree.

By applying strategy DGN, the utilities of the nodes that are closer to the root of a global UP-Tree are further reduced. DGN is especially suitable for the databases containing lots of long transactions. In other words, the more items a transaction contains, the more utilities can be discarded by DGN. On the contrary, traditional TWU mining model is not suitable for such databases since the more items a transaction contains, the higher TWU is. In following sections, we describe the process of constructing a global UP-Tree with strategies DGU and DGN.

3.1.4 Constructing a Global UP-Tree by Applying DGU and DGN

Recall that the construction of a global UP-Tree is performed with two database scans. In the first scan, each transaction's TU is computed; at the same time, each 1-item's TWU is also accumulated. Thus, we can get promising items and unpromising items. After getting all promising items, DGU is applied. The transactions are reorganized by pruning the unpromising items and sorting the remaining promising items in a fixed order. Any ordering can be used such as the lexicographic, support, or TWU order. Each transaction after the above reorganization is called a *reorganized transaction.* In the following paragraphs, we use the TWU descending order to explain the whole process since it is mentioned that the performance of this order is the best in previous study [3].

TABLE 3
Reorganized Transactions and Their RTUs

TID	Reorganized transaction	RTU
T_1'	(C,10) (D,1) (A,1)	17
T_2'	(E,2) (C,6) (A,2)	22
T_3'	(E,2) (D,6) (A,2) (B,2)	32
T_4'	(E,1) (C,13) (D,3) (B,4)	30
T_5'	(E,1) (C,4) (B,2)	11
T_6'	(C,1) (D,1) (A,1) (B,1)	10

Then a function *Insert_Reorganized_Transaction* is called to apply DGN during constructing a global UP-Tree. Its subroutine is shown in Fig. 2. When a reorganized transaction $t_j' = \{i_1, i_2, \ldots, i_n\}(i_k \in I, 1 \leq k \leq n)$ is inserted into a global UP-Tree, *Insert_Reorganized_Transaction* (N, i_x) is called, where N is a node in UP-Tree and i_x is an item in $t_j'(i_x \in t_j', 1 \leq x \leq n)$. First, (N_R, i_1) is taken as input, where N_R is the root node of UP-Tree. The node for i_1, N_{i_1}, is found or created under N_R and its support is updated in Line 1.Then DGN is applied in Line 2 by discarding the utilities of descendant nodes under N_{i_1}, i.e., N_{i_2} to N_{i_n}. Finally, in Line 3, (N_{i_1}, i_2) is taken as input recursively.

An example is given to explain how to apply the two strategies during the construction of a global UP-Tree. Consider the transaction database in Table 1 and the profit table in Table 2. Suppose *min_util* is 50. In the first scan of database, TUs of all transactions and TWUs of distinct items are computed. Five promising items, i.e., $\{\mathrm{A}\} : 93, \{\mathrm{B}\} : 92, \{\mathrm{C}\} : 99, \{\mathrm{D}\} : 96$ and $\{\mathrm{E}\} : 107$, are sorted in the header table by the descending order of TWU, that is, {E}, {C}, {D}, {A}, and {B}. Then the transactions are reorganized by sorting promising items and subtracting utilities of unpromising items from their TUs. The reorganized transactions and their RTUs are shown in Table 3. Comparing Tables 3 and 1, the RTUs of T_2, T_3, and T_5 in Table 3 are less than the TUs in Table 1 since the utilities of {F}, {G}, and {H} have been removed by DGU.

After a transaction has been reorganized, it is inserted into the global UP-Tree. When $\mathrm{T}_1' = \{(\mathrm{C}, 10)(\mathrm{D}, 1)(\mathrm{A}, 1)\}$ is inserted, the first node N_C is created with $\mathrm{N}_\mathrm{C}.item = \{\mathrm{C}\}$ and $\mathrm{N}_\mathrm{C}.count = 1$. N_C.nu is increased by $\mathrm{RTU}(\mathrm{T}_1')$ minus the utilities of the rest items that are behind $\{C\}$ in T_1', that is, $\mathrm{N}_\mathrm{C}.nu = \mathrm{RTU}(\mathrm{T}_1') - (u(\{\mathrm{D}\}, \mathrm{T}_1') + u(\{\mathrm{A}\}, \mathrm{T}_1')) = 17 - (2 + 5) = 10$. Note that it can also be calculated as the sum of utilities of the items that are before item {D} in T_1', *i.e.*, $\mathrm{N}_\mathrm{C}.nu = u(\{\mathrm{C}\}, \mathrm{T}_1') = 10$. The second node N_D is created with $\mathrm{N}_\mathrm{D}.item = \{\mathrm{D}\}, N_\mathrm{D}.count = 1$ and $\mathrm{N}_\mathrm{D}.nu = \mathrm{RTU}(\mathrm{T}_1')$-$u(\{\mathrm{A}\}, \mathrm{T}_1') = 17 - 5 = 12$. The third node N_A is created with $\mathrm{N}_\mathrm{A}.item = \{\mathrm{A}\}, \mathrm{N}_\mathrm{A}.count = 1$ and $\mathrm{N}_\mathrm{A}.\mathrm{nu} = \mathrm{RTU}(\mathrm{T}_1') = 17$.

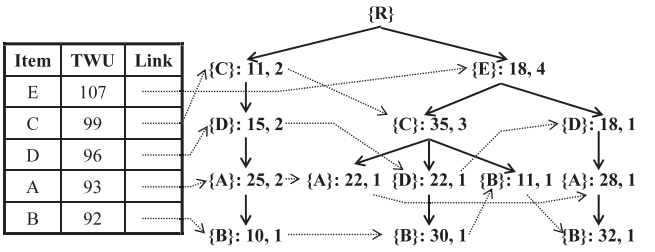


Fig. 3. A UP-Tree by applying strategies DGU and DGN.

After inserting all reorganized transactions by the same way, the global UP-Tree shown in Fig. 3 is constructed. Comparing with the IHUP-Tree in Fig. 1, node utilities of the nodes in UP-Tree are less than those in IHUP-Tree since the node utilities are effectively decreased by the two strategies DGU and DGN.

3.2 The Proposed Mining Method: UP-Growth

After constructing a global UP-Tree, a basic method for generating PHUIs is to mine UP-Tree by FP-Growth [14]. However too many candidates will be generated. Thus, we propose an algorithm *UP-Growth* by pushing two more strategies into the framework of FP-Growth. By the strategies, overestimated utilities of itemsets can be decreased and thus the number of PHUIs can be further reduced. In following sections, we first propose the two strategies and then describe the process of UP-Growth in detail by an example.

3.2.1 Strategy DLU: Discarding Local Unpromising Items during Constructing a Local UP-Tree

The common method for generating patterns in tree-based algorithms [3], [14] contains three steps: 1) Generate *conditional pattern bases* by tracing the paths in the original tree; 2) construct *conditional trees* (also called *local trees* in this paper) by the information in conditional pattern bases; and 3) mine patterns from the conditional trees. However, strategies DGU and DGN cannot be applied into conditional UP-Trees since actual utilities of items in different transactions are not maintained in a global UP-Tree. We cannot know actual utilities of unpromising items that need to be discarded in conditional pattern bases unless an additional database scan is performed.

To overcome this problem, a naïve solution is to maintain items' actual utilities in each transaction into each node of global UP-Tree. However, this is impractical since it needs lots of memory space. In view of this, we propose two strategies, named DLU and DLN, that are applied in the first two mining steps and introduced in this and next sections, respectively. For the two strategies, we maintain a *minimum item utility table* to keep *minimum item utilities* for all global promising items in the database.

Definition 9. Minimum item utility *of item i_p in database D, denoted as $miu(i_p)$, is i_p's utility in transaction T_d if there does not exist a transaction T_d' in D such that $u(i_p, T_d') < u(i_p, T_d)$.*

For example, Table 4 shows the minimum item utility table for global promising items of the database shown in Table 1. Note that minimum item utilities of all items can be collected during the first scan of original database.

TABLE 4
Minimum Item Utility Table

Item	A	B	C	D	E
Minimum item utility	5	2	1	2	3

Minimum item utilities are utilized to reduce utilities of local unpromising items in conditional pattern bases instead of exact utilities. An estimated value for each local unpromising item is subtracted from the *path utility* of an extracted path.

Definition 10. Path utility of *a path p in i_m's conditional pattern base (abbreviated as $\{i_m\}$-CPB) is denoted as $pu(p, \{i_m\}\text{-CPB})$ and defined as N_{i_m}'s node utility where p is retrieved by tracing N_{i_m} in the UP-Tree.*

For example, $pu(<\text{ADC}>, \{\text{B}\}\text{-CPB})$, which is the path utility of the leftest path in Fig. 3 in {B}-CPB, is defined as $N_B.nu$, i.e., 10, in that path. By Definitions 9 and 10, assume that there is a path p in $\{i_m\}$-CPB and $UI_{\{i_m\}\text{-}CPB}$ is the set of unpromising items in $\{i_m\}$-CPB. Path utility of p in $\{i_m\}$-CPB, i.e., $pu(p, \{i_m\}\text{-CPB})$, is recalculated and reduced according to minimum item utilities as below:

$$pu(p, \{i_m\} - \text{CPB}) = N_{i_m}.nu - \sum_{\forall i \in UI_{\{i_m\}\text{-}CPB} \wedge i \subseteq p} miu(i) \times p.count, \quad (1)$$

where $p.\ count$ is the support count of p in $\{i_m\}$-CPB.

Proof. Since UI_{i_m} is the set of unpromising items in $\{i_m\}$-CPB, by Corollary 1, the PHUIs generated from p must not contain the items in UI_{i_m}. Thus, the items in UI_{i_m} and their utilities can be ignored from p. By Definition 9, i_j's actual utility in $\{i_m\}$-CPB (where $i_j \in UI_{i_m}$) must be no less than $miu(i_j) \times p.count$. Thus, p's estimated utility calculated by (1) must always be larger than or equal to its real utility in $\{i_m\}$-CPB. □

By (1), we can define the third proposed strategy for decreasing utilities of local unpromising items in a local UP-Tree as follows:

Strategy 3. DLU: Discarding local unpromising items and their estimated utilities from the paths and path utilities of conditional pattern bases by (1).

DLU can be recognized as local version of DGU. It provides a simple but useful schema to reduce overestimated utilities locally without an extra scan of original database.

3.2.2 Strategy DLN: Decreasing Local Node Utilities during Constructing a Local UP-Tree

As mentioned in the Section 3.1.3, since $\{i_m\}$-Tree must not contain the information about the items below i_m in the original UP-Tree, we can discard the utilities of descendant nodes related to i_m in the original UP-Tree while building $\{i_m\}$-Tree. (Here, *original UP-Tree* means the UP-Tree which is used to generate $\{i_m\}$-Tree.) Because we cannot know

Subroutine : *Insert_Reorganized_Path*(N, i_x)
Line 1 : If N has a child N_{i_x} such that $N_{i_x}.item = i_x$, increment $N_{i_x}.count$ by $p_j.count$. Otherwise, create a new child node N_{i_x} with $N_{i_x}.item = i_x$, $N_{i_x}.count = p_j.count$, $N_{i_x}.parent$ = N and $N_{i_x}.nu = 0$.
Line 2 : Increase $N_{i_x}.nu$ by Eq (3).
Line 3 : If there exists a node N_{i_x} in p_j where $x+1 < m'$, call *Insert_Reorganized_Path*(N_{i_x}, i_{x+1})

Fig. 4. The subroutine of *Insert_Reorganized_Path*.

actual utilities of the descendant nodes, we use minimum item utilities to estimate the discarded utilities.

Definition 11. Path utility of item i_k *in* $\{i_m\}$-CPB *is denoted as* $pu(i_k, \{i_m\}$-$CPB)$ *and defined as the following equation:*

$$pu(i_k, \{i_m\} - \text{CPB}) = \sum_{\forall p \supseteq i_k \wedge p \in \{i_m\}\text{-CPB}} pu(i_k, \{i_m\}\text{-CPB}). \quad (2)$$

Note that the paths discussed here are reorganized by pruning unpromising items by DLU and resorted by a fixed order. The paths are called *reorganized paths*. The reason for forming reorganized paths is the same as forming reorganized transactions when applying DGU. Assume that a reorganized path $p = <N'_{i_1} N'_{i_2} \dots N'_{i'_m}>$ in $\{i_m\}$-CPB is inserted into the path $<N_{i_1} N_{i_2} \dots N_{i'_m}>$ in $\{i_m\}$-Tree, where $m' \leq m$. For the node N_{i_k} in i_m-Tree, where $1 \leq k \leq m'$, $N_{i_k}.nu$ is recalculated as below:

$$N_{i_k}.nu_{new} = N_{i_k}.nu_{old} + pu(p, \{i_m\}CPB) - \sum_{j=k+1}^{m'} miu(i_j) \times p.count, \quad (3)$$

where $N_{i_k}.nu_{old}$ is the node utility of N_{i_k} in $\{i_m\}$-Tree before adding p. (If N_{i_j} to $N_{i_{m'}}$ have not been created in $\{i_m\}$-Tree, node N_{i_k} is created and $N_{i_k}.nu_{old}$ is set as 0.)

Proof. Assume DN_{i_k} stands for the set of descendant nodes below the node N_{i_k} in $\{i_m\}$-Tree. Since the items in DN_{i_k} will not generate PHUIs with i_k in $\{i_k\}$-Tree, their utilities can be discarded from $N_{i_k}.nu$. By Definition 9, we can know i_j's actual utility in p must be no less than $miu(i_j) \times p.count$, where $k+1 \leq j \leq m-1$. Thus, the node utility calculated by (3) must always be larger than or equal to its real utility. □

For example, consider the minimum item utility table in Table 4. Assume a reorganized path <DC> with support count 1 is inserted into a local UP-Tree. The node N_D under root node is created or updated. N_D.nu is increased by $5 - miu(C) \times <DC>.count = 5 - 1 \times 1 = 4$. By (3), we can define the fourth strategy for decreasing utilities of descendant nodes in a local UP-Tree as follows.

Strategy 4. DLN: Decreasing Local Node utilities for the nodes of local UP-Tree by estimated utilities of descendant nodes by (3).

The same as DLU, DLN can be recognized as local version of DGN. By the two strategies, overestimated utilities for itemsets can be locally reduced in a certain degree without losing any actual high utility itemset. The whole process of UP-Growth with DLU and DLN will be addressed in detail in the next section. Then a complete example is given to explain it.

3.2.3 UP-Growth: Mining a UP-Tree by Applying DLU and DLN

The process of mining PHUIs by UP-Growth is described as follows: First, the node links in UP-Tree corresponding to the item i_m, which is the bottom entry in header table, are traced. Found nodes are traced to root of the UP-Tree to get paths related to i_m. All retrieved paths, their path utilities and support counts are collected into i_m's conditional pattern base.

A conditional UP-Tree can be constructed by two scans of a conditional pattern base. For the first scan, local promising and unpromising items are learned by summing the path utility for each item in the conditional pattern base. Then, DLU is applied to reduce overestimated utilities during the second scan of the conditional pattern base. When a path is retrieved, unpromising items and their estimated utilities are eliminated from the path and its path utility by (1). Then the path is reorganized by the descending order of path utility of the items in the conditional pattern base.

DLN is applied during inserting reorganized paths into a conditional UP-Tree. Assume a reorganized path $p_j = <N_{i_1} N_{i_2} \dots N_{i_{m'}}>$, where N_{i_k} is the nodes in UP-Tree and $1 \leq k \leq m'$. When $N_{i_1}.item$, i_1, is inserted into the conditional UP-Tree, the function

$$Insert\_Reorgnized\_Path(N_{R'}, i_1),$$

as shown in Fig. 4, is called, where N_R' is root node of the conditional UP-Tree. The node for i_1, N_{i_1}, is found or created under N_R' and its support is updated in Line 1. Then DLN is applied in Line 2 by decreasing estimated utilities of descendant nodes under N_{i_1}, i.e., N_{i_2} to $N_{i_{m'}}$. Finally in Line 3, (N_{i_1}, i_2) is taken as input recursively.

The complete set of PHUIs is generated by recursively calling the procedure named *UP-Growth*. Initially, *UP-Growth*(T_R, H_R, *null*) is called, where T_R is the global UP-Tree and H_R is the global header table. The procedure of *UP-Growth* is shown in Fig. 5.

Now, we use an example to explain the process of UP-Growth in detail. Consider the UP-Tree in Fig. 3. Suppose *min_util* is 50. The algorithm starts from the bottom entry of header table and considers item {B} first. By tracing all {B}.*hlink*s, sum of {B}'s node utilities is calculated, that is, $nu_{sum}(\{B\}) = 83$. Thus, a new PHUI {B}:83 is generated and {B}-CPB is constructed. By following {B}.*hlink*, the nodes related to {B} are found. By tracing these nodes to root, four paths <ADC> : 10, <DCE> : 30, <CE> : 11, and <ADE> : 32 are found. The number beside a path is path utility of the path, which equals to {B}.nu in each traversed path. These paths are collected into {B}-CPB, which is shown in the first column of Table 5.

Consider {B}-CPB and the minimum item utility table in Table 4. By scanning {B}-CPB once, we can get path utilities of local items in $\{\text{B}\}\text{-CPB} : \{\text{A}\} : 42, \{\text{C}\} : 51, \{\text{D}\} : 72$ and

Subroutine: *UP-Growth*(T_X, H_X, X)
Input: A UP-Tree T_X, a header table H_X for T_X, an itemset X, and a minimum utility threshold *min_util*.
Output: All PHUIs in T_X.

(1) For each entry i_k in H_X do
(2) Trace each node related to i_k via i_k.*hlink* and accumulate $i_k.nu$ to $nu_{sum}(i_k)$;
/* $nu_{sum}(i_k)$: the sum of node utilities of i_k */
(3) If $nu_{sum}(i_k) \geq min\_util$, do
(4) Generate a PHUI $Y = X \bigcup i_k$;
(5) Set $pu(i_k)$ as estimated utility of Y;
(6) Construct Y-CPB;
(7) Put local promising items in Y-CPB into H_Y
(8) Apply DLU to reduce path utilities of the paths;
(9) Apply *Insert_Reorgnized_Path* to insert paths into T_Y with DLN;
(10) If $T_Y \neq$ null then call *UP-Growth*(T_Y, H_Y, Y);
(11) End if
(12)End for

Fig. 5. The subroutine of *UP-Growth*.

$\{E\} : 73$. Thus, a local unpromising item {A} is identified and we can get the decreasing order of path utility of items in {B}-CPB as {E}, {D}, and {C}. During the second scan of {B}-CPB, local unpromising item {A} is removed from the paths <ADC> and <ADE>. At the same time, by DLU, {A}'s estimated utilities in the above two paths are eliminated from path utilities of the two paths by (1), i.e.,

$$pu(\text{<ADC>}, \{\text{B}\}\text{-CPB}) = 10 - miu(\text{A}) \times \text{<ADC>}.count = 10 - 5 \times 1 = 5 \text{ and}$$
$$pu(\text{<ADE>}, \{\text{B}\}\text{-CPB}) = 32 - miu(A) \times \text{<ADE>}.count = 32 - 5 \times 1 = 27.$$

Reorganized paths and their reduced path utilities are shown in second column of Table 5. Comparing the new path utilities in the second column with the old ones in the first column of Table 5, path utilities of the paths are shown to be further reduced after applying strategy DLU.

Now we show the process of constructing a local UP-Tree. Note that the paths are inserted into {B}-Tree simultaneously while they are being reorganized. Consider {B}-CPB shown in Table 5, according to DLN, when the first reorganized path <DC> is inserted into {B}-Tree, the first node N_D is created under root node with $\text{N}_\text{D}.nu = 5 - miu(\text{C}) \times \text{<DC>}.count = 5 - 1 \times 1 = 4$ and $\text{N}_\text{D}.count = 1$. The second node N_C is created under N_D with $\text{N}_\text{C}.nu = 5$ and $\text{N}_\text{C}.count = 1$. After inserting all paths in {B}-CPB, {B}-Tree is constructed completely. Fig. 6a shows the {B}-Tree when DGU, DGN, DLU, and

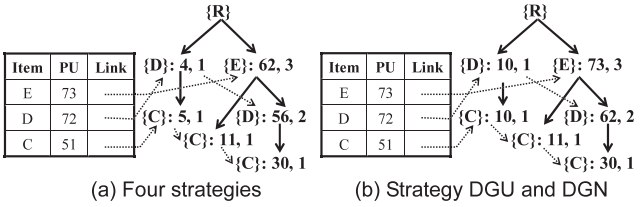


Fig. 6. {B}-Trees with different strategies.

DLN are applied. Comparing the {B}-Tree shown in Fig. 6a with that in Fig. 6b, it can be observed that node utilities of the nodes are further reduced by the strategies DLU and DLN.

Generating PHUIs from {B}-Tree by UP-Growth, the PHUIs that are involved with {B} are obtained, i.e., $\{\text{B}\} : 83$, $\{\text{BD}\} : 60$, $\{\text{BDE}\} : 56$, and $\{\text{BE}\} : 62$. After mining the remaining items in header table, all PHUIs in the UP-Tree in Fig. 3 can be obtained, i.e., $\{\text{A}\} : 75$, $\{\text{B}\} : 83$, $\{\text{BD}\} : 60$, $\{\text{BDE}\} : 56$, $\{\text{BE}\} : 62$, and $\{\text{D}\} : 55$. By applying the four strategies, the generation of PHUIs can be more efficient since the fewer PHUIs are generated, the less time is spent.

3.3 An Improved Mining Method: UP-Growth$^+$

UP-Growth achieves better performance than FP-Growth by using DLU and DLN to decrease overestimated utilities of itemsets. However, the overestimated utilities can be closer to their actual utilities by eliminating the estimated utilities that are closer to actual utilities of unpromising items and descendant nodes. In this section, we propose an improved method, named UP-Growth+, for reducing overestimated utilities more effectively.

In UP-Growth, minimum item utility table is used to reduce the overestimated utilities. In UP-Growth+, *minimal node utilities* in each path are used to make the estimated pruning values closer to real utility values of the pruned items in database.

Definition 12. *Assume that N_x is the node which records the item x in the path p in a UP-Tree and N_x is composed of the items x from the set of transactions $TIDSET(T_X)$. The* minimal node utility *of x in p is denoted as $mnu(x, p)$ and defined as* $\min_{\forall T \in \text{TIDSET}(T_X)}(u(x, T))$.

Minimal node utility for each node can be acquired during the construction of a global UP-Tree. First, we add an element, namely *N.mnu*, into each node of UP-Tree. *N.mnu* is minimal node utility of N. When N is traced, *N.mnu* keeps track of the minimal value of $N.name$'s utility in different transactions. If *N.mnu* is larger than $u(N.name, T_{current})$, $N.mnu$ is set to $u(N.name, T_{current})$. Fig. 7 shows the global UP-Tree with $N.mnu$ in each node. In Fig. 7, *N.mnu* is the last number in each node.

TABLE 5
{B}-CPB After Applying DGU, DGN, and DLU

Retrieved path: Path utility	Reorganized path: Path utility (after DLU)	Support count
<ADC>: 10	<DC>: 5	1
<DCE>: 30	<EDC>: 30	1
<CE>: 11	<EC>: 11	1
<ADE>: 32	<ED>: 27	1

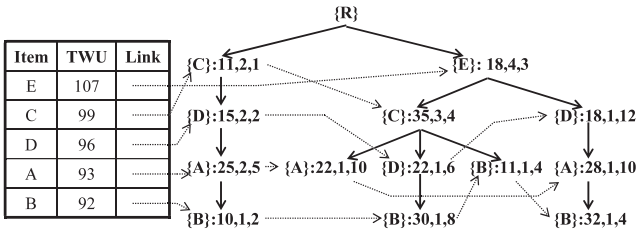


Fig. 7. A UP-Tree with minimal node utilities.

After introducing the modification of global UP-Tree, now we address the processes and two improved strategies of UP-Growth+, named DNU and DNN. When a local UP-Tree is being constructed, minimal node utilities can also be acquired by the same steps of global UP-Tree. In the mining process, when a path is retrieved, minimal node utility of each node in the path is also retrieved. Thus, we can simply replace minimum item utility in (1) and (3) with minimal node utility to obtain the (4) and (5) as follows, respectively.

Assume that there is a path p in $\{i_m\}$-CPB and $UI_{\{i_m\}-CPB}$ is the set of unpromising items in $\{i_m\}$-CPB. The path utility of p in $\{i_m\}$-CPB, i.e., $pu(p, \{i_m\}\text{-CPB})$, is recalculated as below equation:

$$pu(p, \{i_m\}CPB) = p.\{i_m\}.nu - \sum_{\forall i \in UI_{\{i_m\}-CPB} \wedge i \subseteq p} mnu(i) \times p.count, \quad (4)$$

where $p.count$ is the support count of p in $\{i_m\}$-CPB.

Assume that a reorganized path $p = <\mathrm{N'}_{i_1}\mathrm{N'}_{i_2} \ldots \mathrm{N'}_{i_m}>$ in $\{i_m\}$-CPB is inserted into the path $<\mathrm{N}_{i_1}\mathrm{N}_{i_2} \ldots \mathrm{N}_{i_m}>$ in $\{i_m\}$-Tree, where $m' \leq m$. For the node N_{i_k} in $\{i_m\}$-Tree, where $1 \leq k \leq m'$, $\mathrm{N}_{i_k}.nu$ is recalculated as below:

$$\mathrm{N}_{i_k}.nu_{new} = \mathrm{N}_{i_k}.nu_{old} + pu(p, \{i_m\}CPB) - \sum_{j=k+1}^{m'} mnu(i_j) \times p.count, \quad (5)$$

where $\mathrm{N}_{i_k}.nu_{old}$ is the node utility of N_{i_k} in $\{i_m\}$-Tree before adding p.

Since the proof of the above (4) and (5) is similar to that of (1) and (3), it is omitted due to the page limitations. By applying (4) and (5), DNU and DNN can also be defined as the following two strategies.

Strategy 5. DNU. Discarding local unpromising items and their estimated Node Utilities from the paths and path utilities of conditional pattern bases by (4).

Strategy 6. DNN. Decreasing local Node utilities for the nodes of local UP-Tree by estimated utilities of descendant Nodes by (5).

By the above two strategies, the subroutines *Insert_Reorganized_Path* and *UP-Growth* in Figs. 4 and 5 can be modified to the two new subroutines for UP-Growth+. $Insert\_Reorgnized\_Path_{mnu}$ is an improved version of $Insert\_Reorgnized\_Path$ with following modifications. When a new node N_{i_x} is created in Line 1, the element, minimal node utility, is added into N_{i_x} and set $\mathrm{N}_{i_x}.mnu = \infty$ initially. Then, $\mathrm{N}_{i_x}.mnu$ is checked by inserting the procedure "If $\mathrm{N}_{i_x}.mnu > mnu(i_x, p_j)$, set $\mathrm{N}_{i_x}.mnu$ to $mnu(i_x, p_j)$" between Lines 2 and 3 of the subroutine *Insert_Reorgnized_Path.* Finally replace (3) in Line 2 with (5). The modification of the function *UP-Growth* to the new one, named UP-Growth+, is to replace DLU in Line 8 with DNU, and, in Line 9, DLN and *Insert_Reorgnized_Path* with DNN and $Insert\_Reorgnized\_Path_{mnu}$, respectively.

Next, we use an example for describing the processes of UP-Growth+. Consider the UP-Tree in Fig. 7 and assume that *min_util* is set to 50. First, node links of the bottom entry {B} in header table are traced. Four paths are retrieved and added into {B}-CPB : {<A(5)D(2)C(1)> : 10, 1}, {<D(6)C(4)E(3)> : 11, 1}, {<C(4)E(3)> : 30, 1} and {<A(10)D(12)E(3)> : 32, 1}. Note that the number in bracket beside each item is minimal node utility recorded in that node.

TABLE 6
{B}-CPB by Applying DGU, DGN, and DNU

Retrieved path: Path utility	Reorganized path: Path utility (after DNU)	Support count
<A(5)D(2)C(1)>: 10	<D(2)C(1)>: 5	1
<D(6)C(4)E(3)>: 30	<E(3)D(6)C(4)>: 30	1
<C(4)E(3)>: 11	<E(3)C(4)>: 11	1
<A(10)D(12)E(3)>: 32	<E(3)D(12)>: 22	1

By scanning {B}-CPB once, path utility of each local item is calculated: {A}: 42, {C}: 51, {D}: 72, and {E}: 73. According to DNU, local unpromising item {A} and its minimal node utility are discarded from path utilities of the two paths <A(5)D(2)C(1)> and <A(10)D(12)E(3)>. For <A(5)D(2)C(1)>, its path utility is recalculated as $10 - mnu(\mathrm{A}, <\mathrm{ADC}>) \times <\mathrm{ADC}>.count = 10 - 5 \times 1 = 5$; for <A(10)D(12)E(3)>, its path utility is recalculated as $32 - mnu(\mathrm{A}, <\mathrm{ADE}>) \times <\mathrm{ADE}>.count = 32 - 10 \times 1 = 22$. Then the items in each path are reorganized by descending order of the path utility of local items, i.e., {E}, {D}, and {C}, simultaneously. Retrieved paths, reorganized paths and corresponding information of {B}-CPB are shown in Table 6. Comparing Table 6 with Table 5, we can observe the path utility of the last reorganized path <ED> is further reduced by DNU.

When the first reorganized path <D(2)C(1)> is retrieved, according to DNN, the first node N_D is created under the root node with $\mathrm{N_D}.nu = 5 - mnu(\mathrm{C}, <\mathrm{DC}>) \times <\mathrm{DC}>.count = 5 - 1 \times 1 = 4$, $\mathrm{N_D}.count = 1$ and $\mathrm{N_D}.mnu = 2$. The second node $\mathrm{N_C}$ is created under $\mathrm{N_D}$ with $\mathrm{N_C}.nu = 5$, $\mathrm{N_C}.count = 1$, and $N_C.mnu = 1$. The second reorganized path <E(3)D(6)C(4)> is inserted into {B}-Tree by the same process as the first path. New nodes are created sequentially as a new path in {B}-Tree: Node N_E with $\mathrm{N_E}.nu = 30 - (mnu(\mathrm{D}, <\mathrm{EDC}>) \times <\mathrm{EDC}>.count + mnu(\mathrm{C}, <\mathrm{EDC}>) \times <\mathrm{EDC}>.count) = 30 - (6 \times 1 + 4 \times 1) = 20$, $\mathrm{N_E}.count = 1$ and $N_E.mnu = 3$; $\mathrm{N_D}$ with $\mathrm{N_D}.nu = 30 - mnu(\mathrm{C}, <\mathrm{EDC}>) \times <\mathrm{EDC}>.count = 30 - 4 \times 1 = 26$, $\mathrm{N_D}.count = 1$ and $\mathrm{N_D}.mnu = 6$; N_C with $\mathrm{N_C}.nu = 30$, $\mathrm{N_C}.count = 1$, and $\mathrm{N_C}.mnu = 4$.

When the third path <E(3)C(4)> is inserted, $\mathrm{N_E}.nu$ is increased by $20 + (11 - mnu(\mathrm{C}, <\mathrm{EC}>) \times <\mathrm{EC}>.count) = 20 + (11 - 4 \times 1) = 27$ and $\mathrm{N_E}.count$ is increased by 1. Since $\mathrm{N_E}.mnu$ is the same as $mnu(\mathrm{E}, <\mathrm{EC}>)$, i.e., 3, $\mathrm{N_E}.mnu$ is not updated. The second node $\mathrm{N_C}$ is created under $\mathrm{N_E}$ with $\mathrm{N_C}.nu = 11$, $\mathrm{N_C}.count = 1$, and $\mathrm{N_C}.mnu = 4$. After inserting all paths, {B}-Tree shown in Fig. 8 is constructed. Comparing Fig. 8 with Fig. 6a, node utilities in the {B}-Tree in Fig. 8 is further reduced by DNU and DNN.

After mining the whole UP-Tree by UP-Growth+, we can obtain all PHUIs, i.e., {A}:75, {B}:83, and {D}:55 in the UP-Tree. In this example, the number of PHUIs of UP-Growth+ is less than that of UP-Growth. It means that the number of PHUIs, as well as the overestimated utilities of itemsets, are further reduced by UP-Growth+.

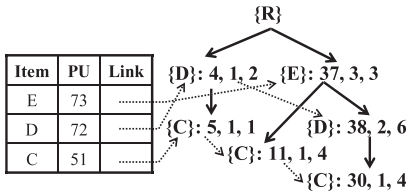


Fig. 8. {B}-Tree by applying DNU and DNN.

3.4 Efficiently Identify High Utility Itemsets

After finding all PHUIs, the third step is to identify high utility itemsets and their utilities from the set of PHUIs by scanning original database once. This step is called *phase II* [3], [19]. However, in previous studies [3], [19], two problems in this phase occur: 1) number of HTWUIs is too large; and (2) scanning original database is very time-consuming. In our framework, overestimated utilities of PHUIs are smaller than or equal to TWUs of HTWUIs since they are reduced by the proposed strategies. Thus, the number of PHUIs is much smaller than that of HTWUIs. Therefore, in phase II, our method is much efficient than the previous methods.

Moreover, although our methods generate fewer candidates in phase I, scanning original database is still time consuming since the original database is large and it contains lots of unpromising items. In view of this, in our framework, high utility itemsets can be identified by scanning reorganized transactions. Since there is no unpromising item in the reorganized transactions, I/O cost and execution time for phase II can be further reduced. This technique works well especially when the original database contains lots of unpromising items.

4 Experimental Evaluation

Performance of the proposed algorithms is evaluated in this section. The experiments were performed on a 2.80 GHz Intel Pentium D Processor with 3.5 GB memory. The operating system is Microsoft Windows 7. The algorithms are implemented in Java language. Both real and synthetic data sets are used in the experiments. Synthetic data sets were generated from the data generator in [1]. Parameter descriptions and default values of synthetic data sets are shown in Table 7. Real world data sets Accidents and Chess are obtained from FIMI Repository [41]; Chain-store is obtained from NU-MineBench 2.0 [23]; Foodmart is acquired from Microsoft foodmart 2000 database. Table 8 shows characteristics of the above data sets. In the above data sets, except Chain-store and Foodmart, unit profits for items in utility tables are generated between 1 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 10. The two real data sets Chain-store and Foodmart already contain unit profits and purchased quantities. Total utilities of the two data sets are 26,388,499.8 and 120,160.84, respectively.

To show the performance of the proposed algorithms, we compared several compared methods and give them new notations as follows: $IHUP_{TWU}$ algorithm, which is proposed in [3] and composed of $IHUP_{TWU}$-Tree and FP-growth, is denoted as IHUPT&FPG. For the proposed

TABLE 7
Parameter Settings of Synthetic Data Sets

Parameter Descriptions		Default
|D|:	Total number of transactions	100k
T:	Average transaction length	10
|I|:	Number of distinct items	1000
F:	Average size of maximal potential frequent itemsets	6
Q:	Maximum number of purchased items in transactions	10

algorithms, we design two methods UPT&UPG and UPT&UPG+ that are composed of the proposed methods UP-Tree and UP-Growth (with DGU, DGN, DLU, and DLN) and the proposed methods UP-Tree and UP-Growth+ (with DGU, DGN, DNU, and DNN), respectively. To further compare the performance of FP-Tree and UP-Tree, a method called UPT&FPG is also proposed. UPT&FPG generates PHUIs from UP-Tree by FP-Growth directly, in other words, only DGU and DGN are applied. Common settings of the above methods are as follows: First, both UP-Tree and IHUP-Tree are constructed by scanning database twice. Second, items in the transactions are rearranged in descending order of their global TWUs during constructing the trees. Third, at phase II, in order to compare the performance with previous work [3], all algorithms identify high utility itemsets by scanning original databases. For convenience, PHUIs and HTWUIs are both called candidates in our experiments.

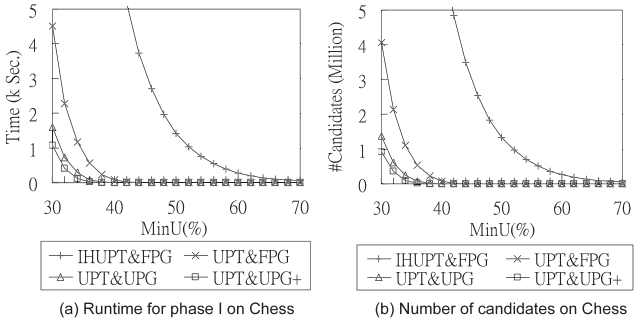
4.1 Performance Comparison on Different Data Sets

In this part, we show the performance comparison on three real data sets: dense data set Chess and sparse data sets Chain-store and Foodmart. First, we show the results on real dense data set Chess in Fig. 9. In Fig. 9a, we can observe that the performance of proposed methods substantially outperforms that of previous methods. The runtime of IHUPT&FPG is the worst, followed by UPT&FPG, UPT&UPG, and UPT&UPG+ is the best. The main reason is the performance of IHUPT&FPG and UPT&FPG is decided by the number of generated candidates. In Fig. 9, runtime of the methods is just proportional to their number of candidates, that is, the more candidates the method produces, the greater its execution time.

Experimental results on real sparse data sets are shown in Fig. 10. The performance on Chain-store data set is shown in Figs. 10a and 10b. In Fig. 10a, the runtime of IHUPT&FPG is the worst, followed by UPT&FPG, UPT&UPG, and UPT&UPG+ is the best. The performance of IHUPT&FPG is the worst since it generates the most

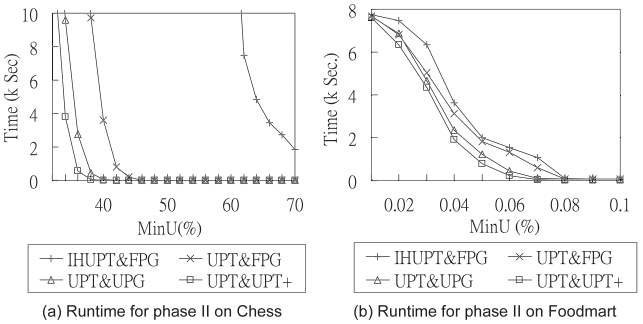
TABLE 8
Characteristics of Real Data Sets

Dataset	|D|	T	|I|	Type
Accidents	340,183	33.8	468	Dense
Chain-store	1,112,949	7.2	46,086	Sparse
Chess	3,196	37.0	75	Dense
Foodmart	4,141	4.4	1,559	Sparse



(a) Runtime for phase I on Chess (b) Number of candidates on Chess

Fig. 9. Performance comparison on dense data set.



(a) Runtime for phase II on Chess (b) Runtime for phase II on Foodmart

Fig. 11. Performance comparison of runtime for Phase II.

candidates. Besides, although the number of candidates of UPT&FPG, UPT&UPG, and UPT&UPG+ are almost the same, the execution time of UPT&FPG is the worst among the three methods since UP-Growth+ and UP-Growth efficiently prune the search space of local UP-Trees.

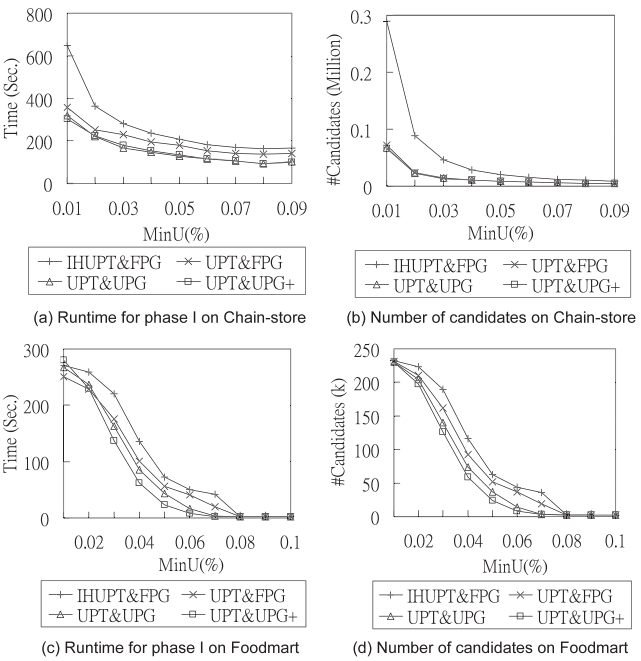
Figs. 10c and 10d show the results on Foodmart data set. In Fig. 10d, number of candidates of the compared methods is almost the same when *min_util* is larger than 0.08 percent or less than 0.01 percent. The reason is that when *min_util* is larger than 0.08 percent, few redundant candidates whose lengths are larger than 2 are generated by IHUP&FPG. On the other hand, when *min_util* is less than 0.01 percent, since it is too small, almost all possible candidates are generated by all compared methods. When *min_util* is between 0.02 and 0.08 percent, the best performer is UPT&UPG+, followed by UPT&UPG, UPT&FPG, and finally IHUPT&FPG. However, when *min_util* is 0.01 percent, the number of candidates is almost the same for each method. The methods needing more calculations consume more execution time, thus, UPT&UPG+ is the worst in this case.

Experimental results of phase II are shown in Fig. 11. We only show the results on Foodmart and Chess since runtime for phase II is very long for large databases, such as Chain-store. In Fig. 11, we can observe that runtime for phase II is not only proportional to number of candidates in phase II but also increases fiercely. Moreover, comparing Figs. 11a and 11b with Figs. 9a and 10c, the runtime of phase II is much more than that of phase I. Such as when *min_util* is 40 percent in Fig. 11a, the runtime for phase II of UPT&FPG is about 3,605 seconds; however, in Fig. 9a, the runtime for phase I of the same method at the same threshold is only 84.15 seconds. Therefore, the performance is highly dependent on the runtime in phase II since the overhead of scanning databases is huge.

By the above results and discussions, we can realize that UP-Tree, UP-Growth, and UP-growth$^+$ efficiently decrease the number of candidates and make the performance much better than that of the state-of-the-art utility mining algorithm IHUP.

4.2 Performance Comparison under Different Parameters

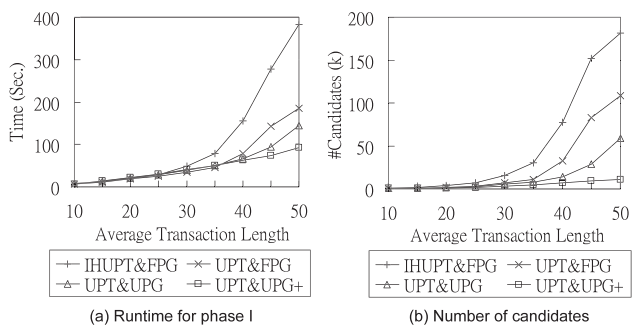
We show the results under different parameters in this part. First, the performance under varied average transaction length (T) is shown in Fig. 12. This experiment is performed on synthetic data sets Tx.F6.|I|1,000.|D|100k and *min_util* is set to 1 percent. In Fig. 12, runtime of all algorithms increases with increasing T because when T is larger, transactions and databases become longer and larger. Also, runtime of the methods is proportional to the number of candidates. The difference of the performance between the methods appears when T is larger than 25. The best method is UPT&UPG+ and the worst one is IHUPT&FPG. In Fig. 12b, the number of candidates generated by UPT&UPG+ is the smallest. This shows that UP-Growth+ can effectively prune more candidates by



(a) Runtime for phase I on Chain-store (b) Number of candidates on Chain-store

(c) Runtime for phase I on Foodmart (d) Number of candidates on Foodmart

Fig. 10. Performance comparison on sparse data sets.



(a) Runtime for phase I (b) Number of candidates

Fig. 12. Varied average transaction length.

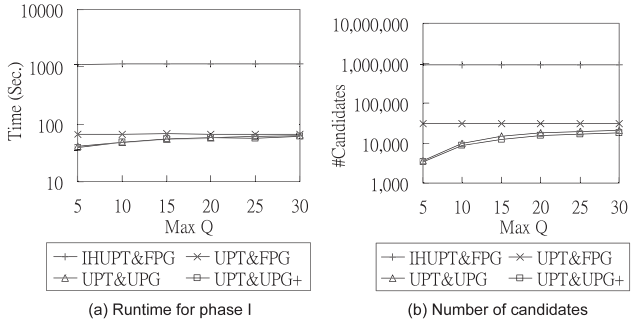


Fig. 13. Varied maximum number of purchased items (accidents).

decreasing overestimated utilities when transactions are longer. In other words, UP-Growth+ is more efficient on the data sets with longer transactions.

Fig. 13 shows the results under varied maximum number of purchased items (Q) on real data set Accidents when *min_util* is set to 20 percent. In Fig. 13a, runtime of the algorithms is proportional to their number of candidates. Overall, the runtime of IHUPT&FPG is the worst, followed by UPT&FPG, with UPT&UPG+ and UPT&UPG being the best. In Fig. 13b, the number of candidates of IHUPT&FPG and UPT&FPG keeps the same with increasing Q; on the other hand, that of UPT&UPG and UPT&UPG+ increases. This is because Q is not a factor of computation in IHUPT&FPG and UPT&FPG. However, UP-Growth and UP-Growth+ use two values, i.e., minimum item utility and minimal node utility, to decrease overestimated utilities. If Q becomes larger, the ratio of the two values to the overestimated utilities will be smaller. In other words, the effect of DLU, DLN, DNU, and DNN will be reduced. Nevertheless, although the number of candidates of UPT&UPG and UPT&UPG+ increases with increasing Q, their upper bound is still the number of candidates of UPT&FPG.

4.3 Performance for Different Sorting Methods

We show the performance about different sorting methods which were mentioned in Section 3.1.4. Fig. 14 shows the results on Accidents data set including following sorting methods on UPT&UPG+: lexicographical order (LEX), support descending order (SUP), TWU descending (TWU(D)), and ascending (TWU(A)) orders, and real utility descending (U(D)) and ascending (U(A)) orders. For U(D) and U(A), we use the order of actual utilities of 1-PHUIs after the first time database scan.

In Fig. 14a, we can observe that the runtime for SUP and TWU(D) are the best and TWU(A) and U(A) are the worst when *min_util* is low. The reasons are shown in Figs. 14b and 14c. Since SUP and TWU(D) generate fewer candidates and construct smaller global UP-Trees, their performances are better. On the other hand, although U(A) generates less candidates, its performance is worse than the other methods except TWU(A) because its UP-Trees are too large. This happens more severely in TWU(A). It even cannot be executed when *min_util* is less than 20 percent because it builds the largest global UP-Trees and runs out of memory for not only larger but also more local UP-Trees. Overall, the support descending

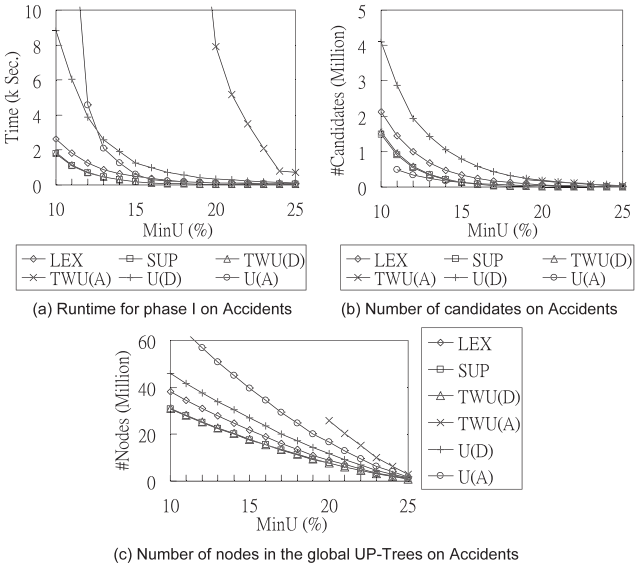


Fig. 14. Performance of UP-Growth+ under different sorting methods.

order is theoretically and consequentially the best since it builds the smallest global UP-Trees and it can prune the largest utility values in UP-Growth+ by DNU and DNN.

4.4 Scalability of the Proposed Methods

In this section, we show the scalability of the compared methods. The experiments are performed on synthetic data sets T10.F6.|I|1,000.|D|xk. Results of runtime for phases I and II, number of candidates and number of high utility itemsets are shown in Fig. 15 and Table 9, respectively. In Fig. 15, we can observe that all compared algorithms have good scalability on runtime. As shown in Fig. 15b, there are only minor differences for runtime of phase I; however, in Fig. 15a, there are significant differences in runtime. Total runtime of UPT&UPG+ is the best, followed by UPT&UPG and UPT&FPG, with IHUPT&FPG being the worst. This is because when the size of database increases, runtime for identifying high utility itemsets also increases. Here, the importance of runtime for phase II is emphasized again. In Table 9, number of PHUIs generated by UP&UPG+ outperforms other methods in the databases with varied database sizes. Overall, the performance of UPG&UPG+ outperforms the other compared algorithms with increasing size of databases since it generates the least PHUIs in phase I.

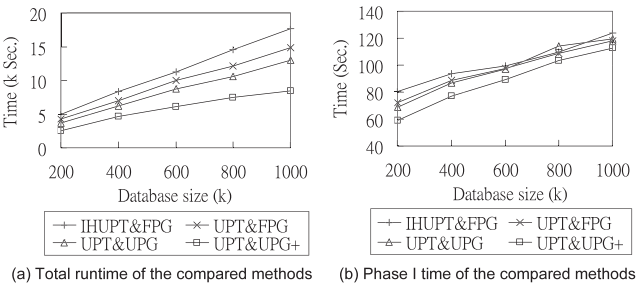


Fig. 15. Experimental results under varied database size.

TABLE 9
Number of Candidates and High Utility Itemsets under Varied Database Sizes

Database	IHUP& FPG	UP& FPG	UP& UPG	UP& UPG+	#HUIs
200k	70,976	60,254	53,844	36,292	8,175
400k	74,551	64,560	58,050	40,556	8,976
600k	69,015	59,918	53,829	37,169	7,324
800k	68,088	59,528	53,491	38,887	8,188
1,000k	68,920	58,160	52,073	36,104	7,144

4.5 Memory Usage of the Proposed Methods

In this part, we discuss memory consumption of the compared methods. Tables 10 and 11 show memory usage of the compared methods (in GB) under varied *min_util* on Chain-store data set and varied database sizes on synthetic data sets T10.F6.|I|1,000.|D|xk, respectively. In Table 10, memory usage of all methods increases with decreasing *min_util* since less *min_util* makes IHUP-Trees and UP-Trees larger. On the other hand, memory usage increases with increasing database size in Table 11. Generally, UP&UPG uses the least memory among the three methods. This is because the strategies effectively decrease the number of PHUIs and local UP-Trees. Besides, if UP&UPG and UP&UPG+ generate similar number of PHUIs, UP&-UPG outperforms UP&UPG+ because it does not need to keep minimal node utilities in tree nodes. On the other hand, the fewer PHUIs UP&UPG+ generates, the less memory it consumes. Generally speaking, there is a tradeoff between runtime and memory usage on UP&UPG and UP&UPG+.

4.6 Summary of the Experimental Results

Experimental results in this section show that the proposed methods outperform the state-of-the-art algorithms almost in all cases on both real and synthetic data sets. The reasons are described as follows.

First, node utilities in the nodes of global UP-Tree are much less than TWUs in the nodes of IHUP-Tree since DGU and DGN effectively decrease overestimated utilities during the construction of a global UP-Tree.

Second, UP-growth and UP-Growth+ generate much fewer candidates than FP-growth since DLU, DLN, DNU, and DNN are applied during the construction of local UP-Trees. By the proposed algorithms with the strategies, generation of candidates in phase I can be more efficient since lots of useless candidates are pruned.

TABLE 10
Memory Usage under Varied *min_util* on Chainstore

Min_util	IHUP&FPG	UP&UPG	UP&UPG+
0.1%	1.017	0.748	0.899
0.08%	1.033	0.923	0.922
0.06%	1.132	1.069	1.021
0.04%	1.294	1.188	1.125
0.02%	1.364	1.288	1.374

TABLE 11
Memory Usage under Varied Database Sizes on T10.F6.III1000.|D|xK (*min_util* = 0.1 Percent)

Database	IHUP&FPG	UP&UPG	UP&UPG+
200k	1.324	1.128	1.079
400k	1.400	1.279	1.285
600k	1.490	1.404	1.452
800k	1.585	1.522	1.626
1000k	1.693	1.648	1.837

Third, generally, UP-Growth+ outperforms UP-Growth although they have tradeoffs on memory usage. The reason is that UP-Growth+ utilizes minimal node utilities for further decreasing overestimated utilities of itemsets. Even though it spends time and memory to check and store minimal node utilities, they are more effective especially when there are many longer transactions in databases. In contrast, UP-Growth performs better only when *min_util* is small. This is because when number of candidates of the two algorithms is similar, UP-Growth+ carries more computations and is thus slower.

Finally, high utility itemsets are efficiently identified from the set of PHUIs which is much smaller than HTWUIs generated by IHUP. By the reasons mentioned above, the proposed algorithms UP-Growth and UP-Growth+ achieve better performance than IHUP algorithm.

5 Conclusions

In this paper, we have proposed two efficient algorithms named UP-Growth and UP-Growth+ for mining high utility itemsets from transaction databases. A data structure named UP-Tree was proposed for maintaining the information of high utility itemsets. PHUIs can be efficiently generated from UP-Tree with only two database scans. Moreover, we developed several strategies to decrease overestimated utility and enhance the performance of utility mining. In the experiments, both real and synthetic data sets were used to perform a thorough performance evaluation. Results show that the strategies considerably improved performance by reducing both the search space and the number of candidates. Moreover, the proposed algorithms, especially UP-Growth+, outperform the state-of-the-art algorithms substantially especially when databases contain lots of long transactions or a low minimum utility threshold is used.

Acknowledgments

This research was supported in part by National Science Council, Taiwan, R.O.C., under grant no. NSC101-2221-E-006-255-MY3 and by the US National Science Foundation through grants IIS-0905215, CNS-1115234, IIS-0914934, DBI-0960443, and OISE-1129076, and the US Department of Army through grant W911NF-12-1-0066.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 487-499, 1994.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int'l Conf. Data Eng.*, pp. 3-14, Mar. 1995.

[3] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 21, no. 12, pp. 1708-1721, Dec. 2009.

[4] C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong, "Mining Association Rules with Weighted Items," *Proc. Int'l Database Eng. and Applications Symp. (IDEAS '98)*, pp. 68-77, 1998.

[5] R. Chan, Q. Yang, and Y. Shen, "Mining High Utility Itemsets," *Proc. IEEE Third Int'l Conf. Data Mining*, pp. 19-26, Nov. 2003.

[6] J.H. Chang, "Mining Weighted Sequential Patterns in a Sequence Database with a Time-Interval Weight," *Knowledge-Based Systems*, vol. 24, no. 1, pp. 1-9, 2011.

[7] M.-S. Chen, J.-S. Park, and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 2, pp. 209-221, Mar. 1998.

[8] C. Creighton and S. Hanash, "Mining Gene Expression Databases for Association Rules," *Bioinformatics*, vol. 19, no. 1, pp. 79-86, 2003.

[9] M.Y. Eltabakh, M. Ouzzani, M.A. Khalil, W.G. Aref, and A.K. Elmagarmid, "Incremental Mining for Frequent Patterns in Evolving Time Series Databases," Technical Report CSD TR#08-02, Purdue Univ., 2008.

[10] A. Erwin, R.P. Gopalan, and N.R. Achuthan, "Efficient Mining of High Utility Itemsets from Large Data Sets," *Proc. 12th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 554-561, 2008.

[11] E. Georgii, L. Richter, U. Rückert, and S. Kramer, "Analyzing Microarray Data Using Quantitative Association Rules," *Bioinformatics*, vol. 21, pp. 123-129, 2005.

[12] J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. Int'l Conf. on Data Eng.*, pp. 106-115, 1999.

[13] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," *Proc. 21th Int'l Conf. Very Large Data Bases*, pp. 420-431, Sept. 1995.

[14] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. ACM-SIGMOD Int'l Conf. Management of Data*, pp. 1-12, 2000.

[15] S.C. Lee, J. Paik, J. Ok, I. Song, and U.M. Kim, "Efficient Mining of User Behaviors by Temporal Mobile Access Patterns," *Int'l J. Computer Science Security*, vol. 7, no. 2, pp. 285-291, 2007.

[16] H.F. Li, H.Y. Huang, Y.C. Chen, Y.J. Liu, and S.Y. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," *Proc. IEEE Eighth Int'l Conf. on Data Mining*, pp. 881-886, 2008.

[17] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," *Data and Knowledge Eng.*, vol. 64, no. 1, pp. 198-217, Jan. 2008.

[18] C.H. Lin, D.Y. Chiu, Y.H. Wu, and A.L.P. Chen, "Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window," *Proc. SIAM Int'l Conf. Data Mining (SDM '05)*, 2005.

[19] Y. Liu, W. Liao, and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," *Proc. Utility-Based Data Mining Workshop*, 2005.

[20] R. Martinez, N. Pasquier, and C. Pasquier, "GenMiner: Mining nonredundant Association Rules from Integrated Gene Expression Data and Annotations," *Bioinformatics*, vol. 24, pp. 2643-2644, 2008.

[21] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-Mine: Fast and Space-Preserving Frequent Pattern Mining in Large Databases," *IIE Trans. Inst. of Industrial Engineers*, vol. 39, no. 6, pp. 593-605, June 2007.

[22] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Moal, and M.C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The Prefixspan Approach," *IEEE Trans. Knowledge and Data Eng.*, vol.16, no.10, pp. 1424-1440, Oct. 2004.

[23] J. Pisharath, Y. Liu, B. Ozisikyilmaz, R. Narayanan, W.K. Liao, A. Choudhary, and G. Memik NU-MineBench Version 2.0 Data Set and Technical Report, http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, 2012.

[24] B.-E. Shie, H.-F. Hsiao, V., S. Tseng, and P.S. Yu, "Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments," *Proc. 16th Int'l Conf. DAtabase Systems for Advanced Applications (DASFAA '11)*, vol. 6587/2011, pp. 224-238, 2011.

[25] B.-E. Shie, V.S. Tseng, and P.S. Yu, "Online Mining of Temporal Maximal Utility Itemsets from Data Streams," *Proc. 25th Ann. ACM Symp. Applied Computing*, Mar. 2010.

[26] K. Sun and F. Bai, "Mining Weighted Association Rules without Preassigned Weights," *IEEE Trans. Knowledge and Data Eng.*, vol. 20, no. 4, pp. 489-495, Apr. 2008.

[27] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Efficient Frequent Pattern Mining over Data Streams," *Proc. ACM 17th Conf. Information and Knowledge Management*, 2008.

[28] F. Tao, F. Murtagh, and M. Farid, "Weighted Association Rule Mining Using Weighted Support and Significance Framework," *Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '03)*, pp. 661-666, 2003.

[29] V.S. Tseng, C.J. Chu, and T. Liang, "Efficient Mining of Temporal High Utility Itemsets from Data Streams," *Proc. ACM KDD Workshop Utility-Based Data Mining Workshop (UBDM '06)*, Aug. 2006.

[30] V.S. Tseng, C.-W. Wu, B.-E. Shie, and P.S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," *Proc. 16th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '10)*, pp. 253-262, 2010.

[31] W. Wang, J. Yang, and P. Yu, "Efficient Mining of Weighted Association Rules (WAR)," *Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '00)*, pp. 270-274, 2000.

[32] H. Yao, H.J. Hamilton, and L. Geng, "A Unified Framework for Utility-Based Measures for Mining Itemsets," *Proc. ACM SIGKDD Second Workshop Utility-Based Data Mining*, pp. 28-37, Aug. 2006.

[33] S.J. Yen and Y.S. Lee, "Mining High Utility Quantitative Association Rules." *Proc. Ninth Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK)*, pp. 283-292, Sept. 2007.

[34] S.J. Yen, Y.S. Lee, C.K. Wang, C.W. Wu, and L.-Y. Ouyang, "The Studies of Mining Frequent Patterns Based on Frequent Pattern Tree," *Proc. 13th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD)*, vol. 5476, pp. 232-241, 2009.

[35] C.-H. Yun and M.-S. Chen, "Using Pattern-Join and Purchase-Combination for Mining Web Transaction Patterns in an Electronic Commerce Environment," *Proc. IEEE 24th Ann. Int'l Computer Software and Application Conf.*, pp. 99-104, Oct. 2000.

[36] C.-H. Yun and M.-S. Chen, "Mining Mobile Sequential Patterns in a Mobile Commerce Environment," *IEEE Trans. Systems, Man, and Cybernetics-Part C: Applications and Rev.*, vol. 37, no. 2, pp. 278-295, Mar. 2007.

[37] U. Yun, "An Efficient Mining of Weighted Frequent Patterns with Length Decreasing Support Constraints," *Knowledge-Based Systems*, vol. 21, no. 8, pp. 741-752, Dec. 2008.

[38] U. Yun and J.J. Leggett, "WFIM: Weighted Frequent Itemset Mining with a Weight Range and a Minimum Weight," *Proc. SIAM Int'l Conf. Data Mining (SDM '05)*, pp. 636-640, 2005.

[39] U. Yun and J.J. Leggett, "WIP: Mining Weighted Interesting Patterns with a Strong Weight and/or Support Affinity," *Proc. SIAM Int'l Conf. Data Mining (SDM '06)*, pp. 623-627, Apr. 2006.

[40] M.J. Zaki, "Scalable Algorithms for Association Mining," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 3, pp. 372-390, May 2000.

[41] Frequent Itemset Mining Implementations Repository, http://fimi.cs.helsinki.fi/, 2012.

Vincent S. Tseng received the PhD degree in computer science from National Chiao Tung University, Taiwan, in 1997. He is a professor in the Department of Computer Science and Information Engineering at National Cheng Kung University (NCKU), Taiwan, ROC. Before this, he was a postdoctoral research fellow in the Computer Science Division at the University of California at Berkeley from January 1998 and July 1999. He has served as the president of the Taiwanese Association for Artificial Intelligence during 2011-2012 and acted as the director for the Institute of Medical Informatics of NCKU during August 2008 and July 2011. He has a wide variety of research interests covering data mining, biomedical informatics, mobile and Web technologies, and multimedia databases. He has published more than 250 research papers in referred journals and conferences and also held (or filed) more than 15 patents. He is an honorary member of Phi Tau Phi Society and has served as chairs/program committee for a number of premier international conferences like SIGKDD, ICDM, PAKDD, CIKM, and IJCAI. He is on the editorial baord of a number of journals including the *IEEE Transactions on Knowledge and Data Engineering* and the *ACM Transactions on Knowledge Discovery from Data*.

Bai-En Shie received the master's degree in computer science and information engineering from Ming Chuan University, Taiwan, R.O.C., in 2006. He is currently working toward the PhD degree in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, R.O.C. His research interests include data mining algorithm, mobile data mining, healthcare system, and temporal data mining.

Cheng Wei Wu received the MS degree in computer science and information engineering from Ming Chuan University, Taiwan, R.O.C., in 2009. He is currently working toward the PhD degree in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, R.O.C. His research interests include data mining and its applications.

Philip S. Yu received the BS degree in electrical engineering from National Taiwan University, the MS and PhD degrees in electrical engineering from Stanford University, and the MBA degree from New York University. He is a professor in the Department of Computer Science at the University of Illinois at Chicago and also holds the Wexler Chair in Information Technology. He spent most of his career at the IBM Thomas J. Watson Research Center and was manager of the Software Tools and Techniques group. His research interests include data mining, database systems, and privacy. He has published more than 560 papers in refereed journals and conferences. He holds or has applied for more than 300 US patents. He is a fellow of the ACM and the IEEE. He is an associate editor of the *ACM Transactions on the Internet Technology* and the *ACM Transactions on Knowledge Discovery from Data*. He is on the steering committee of the IEEE Conference on Data Mining and was a member of the IEEE Data Engineering steering committee. He was the editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering* (2001-2004). He had received several IBM honors including two IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, two Research Division Awards, and the 94th plateau of Invention Achievement Awards. He was an IBM master inventor. He received a Research Contributions Award from IEEE International Conference on Data Mining in 2003 and also an IEEE Region 1 Award for "promoting and perpetuating numerous new electrical engineering concepts" in 1999.