UP-Growth: An Efficient Algorithm for High Utility Itemset Mining

Vincent S. Tseng[1], Cheng-Wei Wu[1], Bai-En Shie[1], and Philip S. Yu[2]

[1] Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC
[2] Department of Computer Science, University of Illinois at Chicago, Chicago, Illinois, USA

tsengsm@mail.ncku.edu.tw, {silvemoonfox, brian0326}@idb.csie.ncku.edu.tw, psyu@cs.uic.edu

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 approaches 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 an efficient algorithm, namely *UP-Growth* (*Utility Pattern Growth*), for mining high utility itemsets with a set of techniques for pruning candidate itemsets. The information of high utility itemsets is maintained in a special data structure named *UP-Tree* (*Utility Pattern Tree*) such that the candidate itemsets can be generated efficiently with only two scans of the database. The performance of UP-Growth was evaluated in comparison with the state-of-the-art algorithms on different types of datasets. The experimental results show that UP-Growth not only reduces the number of candidates effectively but also outperforms other algorithms substantially in terms of execution time, especially when the database contains lots of long transactions.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications — Data Mining.

General Terms

Algorithms, Performance.

Keywords

Utility mining, frequent itemset, high utility itemset, candidate pruning.

1. INTRODUCTION

Frequent itemset mining is a fundamental research topic with wide data mining applications. Extensive studies [1, 5] have been proposed for mining frequent itemsets from the databases and successfully adopted in various application domains. In market analysis, mining frequent itemsets from a transaction database refers to the discovery of the itemsets which frequently appear together in the transactions. However, the unit profits and purchased quantities of items are not considered in the framework of frequent itemset mining. Hence, it cannot satisfy the requirement of the user who is interested in discovering the itemsets with high sales profits. In view of this, utility mining [2, 3, 4, 6, 7, 8, 9, 10] emerges as an important topic in data mining for discovering the itemsets with high utility like profits.

Mining high utility itemsets from the databases refers to finding the itemsets with high utilities. The basic meaning of utility is the interestedness/importance/profitability of items to the users. The 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 the items in the transaction, which is called internal utility. The utility of an itemset is defined as the external utility multiplied by the internal utility. An itemset is called a *high utility itemset* if its utility is no less than a user-specified threshold; otherwise, the itemset is called a *low utility itemset*. Mining high utility itemsets from databases is an important task which is essential to a wide range of applications such as website click streaming analysis, cross-marketing in retail stores, business promotion in chain hypermarkets and even biomedical applications.

However, mining high utility itemsets from the databases is not an easy task since the downward closure property [1] used in frequent itemset mining cannot be applied here. 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 approach for this problem is to enumerate all itemsets from the databases by the principle of exhaustion. Obviously, this approach will encounter the large search space problem, especially when databases contain lots of long transactions or a low threshold is set. Hence, how to effectively prune the search space and efficiently capture all high utility itemsets with no miss is a big challenge in utility mining.

Existing studies [2, 4, 6, 7, 9] applied overestimated methods to facilitate the mining performance of utility mining. In these methods, potential high utility itemsets are found first, and then an additional database scan is performed for identifying their utilities. However, the existing methods often generate a huge set of potential high utility itemsets and the mining performance is degraded consequently. The situation may become worse when the database contains many long transactions or low threshold is set. The huge number of potential high utility itemsets forms a challenging problem to the mining performance since the higher processing cost is incurred with more potential high utility itemsets are generated.

To address this issue, we propose in this paper a novel algorithm with a compact data structure for efficiently discovering high utility itemsets from transactional databases. The major contributions of this work are summarized as follows:

1. A novel algorithm, called *UP-Growth* (*Utility Pattern Growth*), is proposed for discovering high utility itemsets. Correspondingly, a compact tree structure, called *UP-Tree* (*Utility Pattern Tree*), is proposed to maintain the important information of the transaction database related to the utility patterns. High utility itemsets are

then generated from the UP-Tree efficiently with only two scans of the database.

2. Four strategies are proposed for efficient construction of UP-Tree and the processing in UP-Growth. By these strategies, the estimated utilities of candidates can be well reduced by discarding the utilities of the items which are impossible to be high utility or not involved in the search space. The proposed strategies can not only efficiently decrease the estimated utilities of the potential high utility itemsets but also effectively reduce the number of candidates.

3. Both of synthetic and real datasets are used in experimental evaluations to compare the performance of UP-Growth with the state-of-the-art utility mining algorithms. The experimental results show that UP-Growth outperforms other algorithms substantially in terms of execution time, especially when the database contains lots of long transactions.

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 algorithm and data structure are described in details. Experiment results are shown in section 4 and the conclusions are given in section 5.

2. BACKGROUND

In this section, we first define the problem of utility mining and then describe the previous works of utility mining.

2.1 Problem Definition

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 $p(i_p)$. An itemset X is a set of k distinct items $\{i_1, i_2, \ldots, i_k\}$, where $i_j \in I$, $1 \le j \le k$, and k is the length of X. An itemset with length k is called 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 an unique identifier d, called *TID*. Each item i_p in the transaction T_d is associated with a quantity $q(i_p, T_d)$, that is, the purchased number of i_p in T_d.

Definition 1. The utility of an item i_p in the transaction T_d is denoted as $u(i_p, T_d)$ and defined as $p(i_p) \times q(i_p, T_d)$. For example, in Table 1, $u(\{A\}, T_1) = 5 \times 1 = 5$.

Definition 2. The utility of an itemset X in T_d is denoted as $u(X, T_d)$ and defined as $\sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d)$. For example, $u(\{AC\}, T_1) = u(\{A\}, T_1) + u(\{C\}, T_1) = 5 + 1 = 6$.

Definition 3. The utility of an itemset X in D is denoted as $u(X)$ and defined as $\sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$. For example, $u(\{AD\}) = u(\{AD\}, T_1) + u(\{AD\}, T_3) = 7 + 17 = 24$.

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*.

Table 1. An example database

TID	Transaction	TU
T_1	(A,1) (C,1) (D,1)	8
T_2	(A,2) (C,6) (E,2) (G,5)	27
T_3	(A,1) (B,2) (C,1) (D,6) (E,1) (F,5)	30
T_4	(B,4) (C,3) (D,3) (E,1)	20
T_5	(B,2) (C,2) (E,1) (G,2)	11

Table 2. Profit table

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

Problem Statement. Given a transaction database D and a user-specified minimum utility threshold *min_util*, mining high utility itemsets from the transaction database is equivalent to discover from D all itemsets whose utilities are no less than *min_util*.

After addressing the problem definition of utility mining, we introduce the transaction-weighted downward closure which is proposed in [7].

Definition 5. The transaction utility of a transaction T_d is denoted as $TU(T_d)$ and defined as $u(T_d, T_d)$. For example, $TU(T_1) = u(\{ACD\}, T_1) = 8$.

Definition 6. The transaction-weighted utilization 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 $\sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$. For example, $TWU(\{AD\}) = TU(T_1) + TU(T_3) = 8 + 30 = 38$. If *TWU(X)* is no less than the minimum utility threshold, X is called a *high transaction-weighted utilization itemset* (abbreviated as *HTWUI*).

Definition 7. The *transaction-weighted downward closure*, which is abbreviated as *TWDC*, is stated as follows. For any itemset X, if X is not a HTWUI, any superset of X is a low utility itemset. By this definition, the *downward closure property* can be maintained by using transaction-weighted utilization. For example, in Table 1, any superset of $\{AD\}$ is a low utility itemset since $TWU(\{AD\}) <$ *min_util*.

2.2 Related Work

Extensive studies have been proposed for mining frequent itemsets. One of the well-known algorithms is Apriori algorithm [1], which is the pioneer for efficiently mining association rules from large databases. The tree-based approaches such as FP-Growth [5] were afterward proposed. It's widely recognized that FP-Growth achieves a better performance than Apriori-based approaches since it finds frequent itemsets without generating any candidate itemset and it scans database just twice.

However, in the framework of frequent itemset mining [1, 5], the importance of items to users is not considered. The unit profits and purchased quantities of the items are not taken into considerations. Thus, some methods were proposed for mining high utility itemsets from the databases, such as UMining [9], Two-Phase [7], IIDS [6] and IHUP [2]. UMining algorithm [9] proposed by Yao et al. used an estimation method to prune search space. Although it is shown to have good performance, it cannot capture the complete set of high utility itemsets since some high utility patterns may be pruned during the process.

Two-Phase algorithm [7] proposed by Liu et al. consists of two phases. In phase I, Two-Phase algorithm employs a breadth first search strategy to enumerate HTWUIs. It generates candidate itemsets of length k from HTWUIs of length (k-1) and prunes candidate itemsets by TWDC property. In each pass, HTWUIs and their estimated utility values i.e., TWUs, are computed by scanning database. After that, the complete set of HTWUIs is collected in phase I. In phase II, high utility itemsets and their

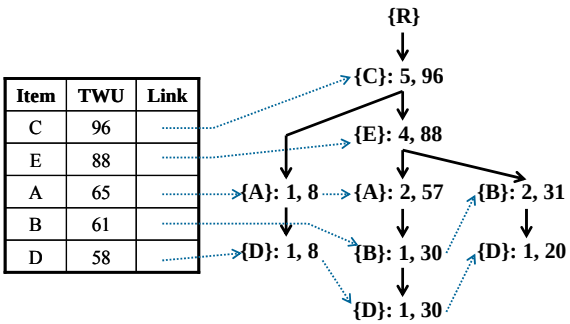
utilities are identified from the HTWUIs by scanning original database once.

Although Two-Phase algorithm effectively reduces the search space by TWDC property and captures the complete set of high utility itemsets, it still generates too many candidates for HTWUIs and requires multiple database scans. To overcome this problem, Li et al. [6] proposed an isolated items discarding strategy, abbreviated as IIDS, to reduce the number of candidates. By pruning isolated items during the level-wise search, the number of candidate itemsets for HTWUIs in phase I can be reduced effectively. However, this approach still scans database multiple 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 multiple times, Ahmed et al. [2] proposed a tree-based algorithm, called IHUP, for mining high utility itemsets. They use an IHUP-Tree to maintain the information of high utility itemsets and transactions. Every node in IHUP-Tree consists of an item name, a support count, and a TWU value. The framework of the algorithm consists of three steps: (1) The construction of IHUP-Tree, (2) the generation of HTWUIs and (3) identification of high utility itemsets. The phase I of IHUP In step 1, items in the transaction are rearranged in a fixed order such as lexicographic order, support descending order or TWU descending order. Then, the rearranged transactions are inserted into the IHUP-Tree. Figure 1 shows a global IHUP-Tree for the database in Table 1, in which items are arranged in the descending order of TWU. In step 2, HTWUIs are generated from the IHUP-Tree by applying the FP-Growth algorithm [5]. Thus, HTWUIs in phase I can be found more efficiently without generating candidates 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 finds HTWUIs without generating any candidates for HTWUIs and 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 use transaction-weighted utilization mining model [7] to overestimate the utilities of the itemsets. However, this model may overestimate too many low utility itemsets as HTWUIs and produce too many candidate itemsets in phase I. Such a large number of HTWUIs degrades the mining performance in phase I in terms of execution time and memory consumption. Besides, the number of HTWUIs in phase I also affects the performance of the algorithms in phase II since the more HTWUIs are generated in phase I, the more execution time is required for identifying high utility itemsets in phase II.

As stated above, the number of HTWUIs generated in phase I forms a crucial problem to the performance of algorithms. In view of this, we propose four strategies to reduce the estimated utility values of the itemsets. By applying the proposed strategies, the number of candidates generated in phase I can be reduced



Item	TWU	Link
C	96	
E	88	
A	65	
B	61	
D	58	

Figure 1: An IHUP-Tree when *min_util* = 40.

effectively and the high utility itemsets can be identified more efficiently since the number of itemsets needed to be checked in phase II is highly reduced in phase I.

3. PROPOSED METHOD

In this section, we first introduce the proposed data structure, named *UP-Tree*, and then describe the proposed algorithm, called *UP-Growth*, in details. The framework of the proposed method consists of three parts: (1) construction of UP-Tree, (2) generation of potential high utility itemsets from the UP-Tree by UP-Growth, and (3) identification of high utility itemsets from the set of potential high utility itemsets. Note that we use a new term *potential high utility itemsets* (*PHUIs*) to distinguish the discovered patterns found by our approach from the HTWUIs since our approach is not based on the traditional framework of transaction-weighted utilization mining model. In our proposed model, the set of PHUIs is much smaller than the set of HTWUIs in phase I.

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, called *UP-Tree* to maintain the information of transactions and high utility itemsets.

3.1.1 The elements in UP-Tree

In UP-Tree, each node *N* includes *N.name*, *N.count*, *N.nu*, *N.parent*, *N.hlink* and a set of child nodes. The details are introduced as follows. *N.name* is the item name of the node. *N.count* is the support count of the node [5]. *N.nu* is called node utility which is an estimate utility value of the node. *N.parent* records the parent node of the node. *N.hlink* is a node link which points to a node whose item name is the same as *N.name*.

Header table is employed to facilitate the traversal of UP-Tree. In the header table, each entry is composed of an item name, an estimate utility value, 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 link in the header table and the nodes in UP-Tree, the nodes whose item names are the same can be traversed efficiently.

3.1.2 Discarding global unpromising items during the construction of a global UP-Tree

The construction of UP-Tree can be performed with two scans of the original database. In the first scan of database, the transaction utility of each transaction is computed. At the same time, TWU of each single item is also accumulated. After scanning database

Table 3. Items and their TWUs

Item	A	B	C	D	E	F	G
TWU	65	61	96	58	88	30	38

Table 4. Reorganized transactions and their RTUs

TID	Reorganized transaction	RTU
T_1'	(C,1) (A,1) (D,1)	8
T_2'	(C,6) (E,2) (A,2)	**22**
T_3'	(C,1) (E,1) (A,1) (B,2) (D,6)	**25**
T_4'	(C,3) (E,1) (B,4) (D,3)	20
T_5'	(C,2) (E,1) (B,2)	**9**

once, items and their TWUs are obtained. By TWDC property, if the TWU of an item is less than minimum utility threshold, its supersets are unpromising to be high utility itemsets. The item is called *unpromising items*. Definition 8 gives a formal definition for unpromising items and promising items.

Definition 8. (Promising item and unpromising item) An item i_p is called a *promising item* if $TWU(i_p) \geq min\_util$. Otherwise, the item is called an *unpromising item*.

After the first scan of database, promising items are organized in the header table in the descending order of TWU values. Note that other orders can be used. In this paper, we suggest the TWU descending order since [2] indicates that this order facilitates the mining performance. During the second scan of database, transactions are inserted into UP-Tree. Initially, the tree is created with a root *R*. When a transaction is retrieved, unpromising items are removed from the transaction and their utilities are eliminated from the TU of the transaction since only the supersets of promising items are possible to be the high utility itemsets. The remaining promising items in the transaction are sorted in the descending order of TWU. The transaction after the above reorganization is called *reorganized transaction* and its TU is called *RTU* (*reorganized transaction utility*). The RTU of a reorganized transaction T_d is denoted as $RTU(T_d)$.

Example 1. Consider the transaction database in Table 1 and the profit table in Table 2. Suppose the minimum utility threshold *min_util* is 40. In the first scan of database, TUs of the transactions and the TWUs of the items are computed. They are shown in the last column of Table 1 and in Table 3, respectively. As shown in Table 3, {*F*} and {*G*} are unpromising items since their TWUs are less than *min_util*. The promising items are reorganized in the header table in the descending order of TWU. Table 4 shows the reorganized transactions and their RTUs for the database in Table 1. As shown in Table 4, unpromising items {*F*} and {*G*} are removed from the transactions T_2, T_3 and T_5, respectively. Besides, the utilities of {*F*} and {*G*} are eliminated from the TUs of T_2, T_3 and T_5, respectively. The remaining promising items {*A*}, {*B*}, {*C*}, {*D*} and {*E*} in the transaction are sorted in the descending order of TWU. Then, we insert reorganized transactions into the UP-Tree by the same processes as IHUP-Tree [2]. We use the following example to describe the operation of insertion.

Example 2. Consider the reorganized transactions in Table 4. The first reorganized transaction T_1' = {*C*, *A*, *D*} leads to create a branch in UP-Tree. The first node {*C*} is created under the root with {*C*}.*count* = 1 and {*C*}.*nu* = 8. The second node {*A*} is created under node {*A*} with {*A*}.*count* = 1 and {*A*}.*nu* = 8. The third node {*C*} is created as a child of node {*A*} with {*C*}.*count* = 1 and {*C*}.*nu* = 8. When the next reorganized transaction T_2' = {*C*, *E*, *A*} is retrieved, the node utility of the node {*C*} is increased by 22 and {*C*}.*count* is increased by 1. Then, a new node {*E*} is created under {*C*} with {*E*}.*count*=1 and {*E*}.*nu* = 22. Similarly, a new node {*A*} is created under the node {*E*} with {*A*}.*count*=1 and {*A*}.*nu* = 22. The reorganized transactions T_3', T_4' and T_5' are inserted in the same way. After inserting all reorganized transactions, the construction of a global UP-Tree with strategy DGU is complete. The global UP-Tree is shown in Figure 2.

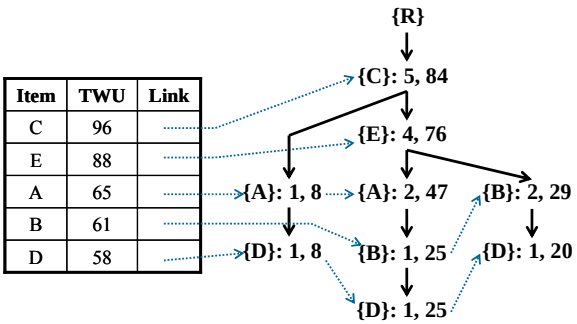


Figure 2. A UP-Tree by applying strategy DGU.

Strategy 1. Discarding global unpromising items (DGU). The unpromising items and their utilities are eliminated from the transaction utilities during the construction of a global UP-Tree.

Rationale: The principle of DGU strategy is to discard the information of unpromising items from the database since an unpromising item plays no role in high utility itemsets and only the supersets of promising items are likely to be high utility.

3.1.3 Generating PHUIs from the global UP-Tree by FP-Growth

In Figure 2, each node in the UP-Tree is associated with two numbers: the first one is support count and the second one is node utility. Besides, the nodes which have the same item names are linked in a sequence by their node links. Comparing with the IHUP-Tree in Figure 1, the node utilities of the nodes in UP-Tree are less than the node utilities of the nodes in IHUP-Tree since reorganized transactions are inserted with RTUs instead of TWUs. In the UP-Tree, each node $\{a_i\}$ to the root forms a path ($\{a_i\} \rightarrow \{a_{i+1}\} \rightarrow ... \rightarrow \{a_n\}$). Each path represents a common prefix that shared by multiple reorganized transactions. Besides, $\{a_i\}.count$ is the number of reorganized transactions that share the path and $\{a_i\}.nu$ is an estimate utility value for the path. Similar to [2], PHUIs can be generated from the UP-Tree by applying FP-Growth [5].

Example 3. Consider the UP-Tree in Figure 2. Suppose *min_util* is 40. The algorithm starts from the bottom of the header table and considers the item {*D*} first. By applying FP-Growth, a PHUI {*D*}:58 is generated since its estimate utility value, i.e., 58, is above than *min_util*. By following {*D*}.*hlink*, the nodes with the same item names are found. By tracing the nodes to root, three paths (D→A→C: 1, 8), (D→B→A→E→C: 1, 25) and (D→B→E→C: 1, 20) are found. For each path, the first number beside the path is the support count and the second number is the path utility, which is equal to {*D*}.*nu*. These paths are collected into {*D*}*'s conditional pattern base* [5] which is denoted as {*D*}*-CPB* and shown in Table 5. In this table, the collected paths are shown in the first column; the support counts and the path

Table 5. {*D*}'s conditional pattern base

Path	Reorganized path	Support count	Path utility by strategies DGU, DGN
{AC}	{C}	1	8
{BAEC}	{CBE}	1	25
{BEC}	{CBE}	1	20

Table 6. Local items and their path utilities in {*D*}-*CPB*

Item	A	B	C	E
Path utility	33	45	53	45

utilities of the paths are shown in the third and the forth columns, respectively. For convenience, the path ($\{a_i\}\rightarrow\{a_{i+1}\}\rightarrow...\rightarrow\{a_n\}$) in the conditional pattern base is denoted as $\{a_i, a_{i+1},..., a_n\}$ and the item a_i is discarded from the path in $\{a_i\}$'s conditional pattern base since every path in $\{a_i\}$-*CPB* must contain a_i.

Definition 9. (Path utility of a path in a conditional pattern base) The path utility of a path $p_j = (\{a_i\}\rightarrow\{a_{i+1}\}\rightarrow...\rightarrow\{a_n\})$ in $\{a_i\}$-*CPB* is equal to $\{a_i\}.nu$ and is denoted as $pu(p_j, \{a_i\}\text{-}CPB)$. For example, in Table 5, the path utility of the path $\{AC\}$ in $\{D\}$-*CPB* is 8.

Definition 10. (Path utility of an item in a path in a conditional pattern base) For each item i_p in the path p_j in $\{a_i\}$-*CPB*, the path utility of an item i_p in a path p_j in $\{a_i\}$-*CPB* is equal to $pu(p_j, \{a_i\}\text{-}CPB)$and denoted as $pu(i_p, p_j)$. For example, the path utility of $\{A\}$ in the path $\{AC\}$ is 8.

Definition 11. (Path utility of an item in a conditional pattern base) The path utility of an item i_p in $\{a_i\}$-*CPB* is defined as $\sum_{i_p\in p_j \wedge p_j\in\{a_i\}-CPB} pu(i_p, p_j)$, which is denoted as $pu(i_p, \{a_i\}\text{-}CPB)$. For example, the path utility of item $\{A\}$ in $\{D\}$-*CPB* is equal to $(pu(\{A\}, \{AC\}) + pu(\{A\}, \{BAEC\})) = (8 + 25) = 33$.

Definition 12. (Local promising item in a conditional pattern base) An item i_p is called a *local promising item* in $\{a_i\}$-*CPB* if $pu(i_p, \{a_i\}\text{-}CPB) \geq min\_util$; otherwise, i_p is called a *local unpromising item*.

Property 1. Let i_u be a *local unpromising item* in $\{a_i\}$-*CPB*, any superset of i_u is not a high utility itemset.

Example 5. By scanning $\{D\}$-*CPB* once, items and their path utilities are obtained, which is shown in Table 6. In Table 6, item $\{A\}$ is a local unpromising item since its path utility is less than *min_util*, i.e., 33<40. Then, local promising items $\{B\}$, $\{C\}$ and $\{E\}$ are arranged in the local header table. Scan $\{D\}$-*CPB* again to construct $\{D\}$*'s conditional UP-Tree* [5], which is denoted as $\{D\}$-*Tree*. When a path in the conditional pattern base is retrieved, unpromising items are removed from the path and the remaining items are rearranged in the descending order according to their local path utilities. The reorganized paths are shown in the second column of Table 5. After inserting all reorganized paths, $\{D\}$-*Tree* is constructed completely and shown in Figure 4(a). Generating PHUIs from $\{D\}$-*Tree* by applying FP-Growth, a set of PHUIs which are involved with item $\{D\}$ are obtained, that is, {{*D*}:58, {*DE*}:45, {*DEB*}:45, {*DEC*}:45, {*DEBC*}:45, {*DB*}:45, {*DBC*}:45, {*DC*}:53}. Consider the next item, i.e., $\{B\}$, in the global header table in the same manner, it derives a set of PHUIs which includes item $\{B\}$, that is, {{*B*}:61 {*BE*}:54, {*BEC*}:54, {BC}:54}. Consider the remaining items in the header table and we can obtain the rest PHUIs, i.e., {{*A*}:65, {*AC*}:55, {*ACE*}:47, {*AE*}:47, {*E*}:88, {*EC*}:76, {*C*}:96}. After finding all PHUIs, high utility itemsets and their utilities are identified from the set of PHUIs by scanning original database once.

As stated above, we have shown a basic framework of our approach for mining high utility itemsets. In the above examples, the strategy DGU is also presented to decrease the estimated utilities of the itemsets. DGU strategy uses RTU to estimate the utilities of the itemsets instead of using TWU. By applying DGU, unpromising items and their utilities are excluded from the UP-Tree such that the node utilities of the nodes are less than the TWUs of the nodes in IHUP-Tree. As a result, the number of PHUIs generated by the proposed approach is less than the HTWUIs generated by IHUP and Two-Phase in phase I. Although DGU seems simple, it is quite effective especially when the transactions contain lots of unpromising items. Besides, DGU can be easily integrated into other TWU-based approaches [2, 7]. Moreover, before the construction of UP-Tree, DGU can be used repeatedly till all reorganized transactions contain no unpromising items. Due to the page limit, we do not discuss it in this paper and leave it in the future work. After addressing DGU, we propose *DGN* (*Decreasing Global Node utilities*) strategy to further reduce the number of PHUIs.

3.1.4 Decreasing node utilities in construction of a global UP-Tree

As shown in example 5, the search space of high utility itemsets can be divided into five smaller search spaces: (1) $\{D\}$-*CPB*, (2) $\{B\}$-*CPB* without containing item $\{D\}$, (3) $\{A\}$-*CPB* without containing items $\{B\}$ and $\{D\}$, (4) $\{E\}$-*CPB* without containing items $\{A\}$, $\{B\}$ and $\{D\}$, and (5) $\{C\}$-*CPB* without containing items $\{E\}$, $\{A\}$, $\{B\}$ and $\{D\}$.

When DGU strategy is applied, there are two paths $\{AEC\}$: 25 and $\{EC\}$: 29 in $\{B\}$-*CPB*, where the numbers beside the paths are their path utilities. Although $\{D\}$ doesn't appear in $\{B\}$'s conditional pattern base, the utility of $\{D\}$ is involved in the path utilities of the paths in $\{B\}$-*CPB*. The path utility of the path $\{AEC\}$ is 25, which is equal to the RTU of the reorganized transaction T_3'. This estimated utility value is actually the sum of $u(\{B\}, T_3')$, $u(\{D\}, T_3')$, $u(\{A\}, T_3')$, $u(\{E\}, T_3')$ and $u(\{C\}, T_3')$. However, all paths in $\{B\}$-*CPB* are not related with $\{D\}$ since $\{D\}$ is below $\{B\}$ in the UP-Tree as shown in Figure 2. Besides, only the item which is an ancestor of the node $\{B\}$ will appear in $\{B\}$-*CPB*. Any item which is a descendant of the node $\{B\}$ will not appear in $\{B\}$-*CPB*. Therefore, the utilities of $\{B\}$'s descendants can be removed from the path utility of each path in $\{B\}$-*CPB*. The process can be done during the construction of global UP-Tree since the paths in the conditional pattern bases are directly derived from the global UP-Tree.

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, the function ***Insert_Reorganized_Transaction***(R, i_1) is called. The function ***Insert_Reorganized_Transaction***(N, i_x) takes a node N in the UP-Tree and an item i_x ($i_x \in t_j'$, $1 \leq x \leq n$) in the reorganized transaction t_j' as inputs. The function is performed as follows:

Line 1: If N has a child S such that $S.item = i_x$, then increment $S.count$ by 1; otherwise, created a new child node S with $S.item = i_x$, $S.count = 1$, $S.parent = N$ and $S.nu = 0$.

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

Line 3: Call ***Insert_Reorganized_Transaction***(S, i_{x+1}) if $p \neq n$.

Example 6. Consider the reorganized transactions in Table 4. When T_1' = {(*C*,1) (*A*,1) (*D*,1)} is inserted to a global UP-Tree, the first node {*C*} is created. {*C*}.*nu* is increased by the RTU of T_1' minus the utilities of the rest items which are behind the item {*C*} in T_1', i.e., {*C*}.*nu* = RTU(T_1') – (u({*A*}, T_1') + u({*D*}, T_1')) = 8 – (5+2) = 1. For convenience, it can also be considered as the sum of the utilities of the items which are before the item {*D*} in T_1', i.e., {*C*}.*nu* = p({*C*}) × q({*C*}, T_1') = 1×1 = 1, where p({*C*}) is the unit profit of the item {*C*} and q({*C*}, T_1') is the purchased number of {*C*} in T_1'. The second node {*A*} is crated with {*A*}.*count* = 1 and {*A*}.*nu* = (p({*C*}) × q({*C*}, T_1') + p({*A*}) × q({*A*}, T_1')) = (1×1 + 5×1) = 6. The third node {*D*} is created with {*D*}.*count* = 1 and {*D*}.*nu* = (p({*C*}) × q({*C*}, T_1') + p({*A*}) × q({*A*}, T_1') + p({*D*}) × q({*D*}, T_1')) = (1×1 + 5×1 + 1×2) = 8. When T_2' = {(C, 6) (E, 2) (A, 2)} is inserted into the tree, {*C*}.*nu* is increased by p({*C*}) × q({*C*}, T_2') = 6 and {*C*}.*count* is increased by 1. Then, a new node {*E*} is created under the node {*C*} with {*E*}.*count* = 1 and {*E*}.*nu* = 12. Similarly, a new node {*A*} is created under the node {*E*} with {*A*}.*count* = 1 and {*A*}.*nu* = 22. After inserting all reorganized transactions, the global UP-Tree is constructed completely. Figure 3 shows the global UP-Tree. By Figure 3, we can know that the node utility of each node is significantly reduced. Generating PHUIs from the UP-Tree by applying FP-Growth, and we obtain a set of PHUIs, that is, {{*D*}:58, {*DE*}:45, {*DEB*}:45, {*DEBC*}:45, {DEC}:45, {*DB*}:45, {*DBC*}:45, {*DC*}:53, {*B*}:61, {*A*}:65, {*E*}:88, {*C*}:96}.

Strategy 2. Discarding global node utilities (DGN). For any node in a global UP-Tree, the utilities of its descendants are discarded from the utility of the node during the construction of a global UP-Tree.

Rationale: Let <i_1, i_2, …, i_n> be a list of promising items which are arranged by the descending order of TWU values in a global header table. Since the items i_{k+1}, i_{k+2}, ..., i_n are not involved in i_k-*CPB* and i_k-*Tree*, they won't be contained in any PHUI of i_k-*Tree*. Thus, their utilities can be discarded from the node i_k in the global UP-Tree.

By applying strategy DGN, the utilities of the nodes which are closer to the root of the global UP-Tree are effectively reduced. DGN strategy is especially suitable for the database which contains lots of long transactions since the more items are in the transactions and the more utilities can be discarded by DGN. On the contrary, traditional transaction-weighted utilization mining model may be unsuitable for the long transactions since the more items in the transaction, the higher TWU is.

3.2 The Proposed Mining Method: UP-Growth

In this section, we describe the details of UP-Growth for efficiently generating PHUIs from the global UP-Tree with two strategies, namely *DLU* (*Discarding local unpromising items*) and *DLN* (*Decreasing local node utilities*). Although strategies DGU and DGN described in previous section can effectively reduce the number of candidates in phase I, they are applied during the construction of the global UP-Tree and cannot be applied during the construction of the local UP-Tree. The reason is that the individual items and their utilities are not maintained in the conditional pattern base. We cannot know the utility values of the unpromising items in the conditional pattern base. To overcome this problem, a naïve approach is to maintain the utilities of the items in the conditional pattern base. However, this approach may

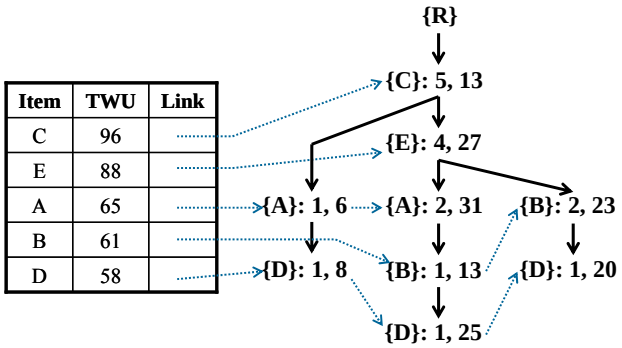


Figure 3. A UP-Tree by applying strategies DGU and DGN.

Table 7. Minimum item utility table

Item	A	B	C	D	E
Minimum item utility	5	4	1	2	3

be impractical since it consumes lots of memory usages. Instead of maintaining exact utility values of the items in the conditional pattern base, we maintain a *minimum item utility table*, abbreviated as *MIUT*, to maintain the minimum item utility for all global promising items.

Definition 13. (**Minimum item utility of an item**) The utility of item i_p in transaction T_d is called the *minimum item utility* of i_p if there doesn't exist a transaction T_d' such that u(i_p, T_d') < u(i_p, T_d). The minimum item utility of i_p is denoted as miu(i_p).

Definition 14. (**Minimum item utility of an item in a path**) The minimum item utility of item i_u in path p_j is defined as miu(i_u) × p_j.*count*, where p_j.*count* is the support count of the path p_j in the conditional pattern base.

Note that the MIUT can be constructed during the first scan of database. Table 7 shows the MIUT for all global promising items in Table 1.

Strategy 3. Discarding local unpromising items (DLU). The minimum item utilities of unpromising items are discarded from path utilities of the paths during the construction of a local UP-Tree.

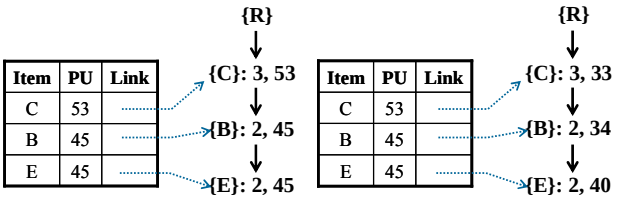
Rationale: By the rationale of DGU strategy, in a conditional pattern tree, local unpromising items and their utilities can be discarded. Since the minimum item utility of a local unpromising item in a path is always equal to or less than its real utility in the path, we can also discard its minimum item utility from the paths of the conditional pattern tree without losing any PHUI.

The purpose of DLU strategy is similar to DGU strategy, while DLU is applied during the second scan of the conditional pattern base. First, we scan conditional pattern base once to identify local promising items and unpromising items. Then, we scan conditional pattern base again to construct a local UP-Tree. When a path is retrieved, each unpromising item is removed from the path and its minimum item utility in this path is eliminated from the path utilities.

Example 7. Consider {*D*}'s conditional pattern base shown in Table 5. Table 6 shows the local items in {*D*}-*CPB* and their path

Table 8. {*D*}-*CPB* by applying DGU, DGN and DLU

Path	Reorganized path	Support count	Path utility by strategies DGU, DGN, DLU
{AC}	{C}	1	3
{BAEC}	{CBE}	1	20
{BEC}	{CBE}	1	20



(a) Strategy DGU (b) Four strategies

Figure 4. {*D*}'s conditional UP-Tree.

utilities. In Table 6, a local unpromising item $\{A\}$ is identified. During the second scan of {*D*}-*CPB*, local unpromising item $\{A\}$ is removed from the path $\{AC\}$ and $\{BAEC\}$, respectively. The minimum item utilities of $\{A\}$ in the above paths, that is, $miu(\{A\})\times \{AC\}.count = 5\times1 = 5$ and $miu(\{A\}) \times \{BAEC\}.count = 5\times1$, are eliminated from the path utilities of $\{AC\}$ and $\{BAEC\}$, respectively. After that, the reorganized paths and the reduced path utilities are shown in Table 8. Here, the path utilities of the paths in {*D*}-*CPB* are shown to be further reduced after applying strategy DLU.

Strategy 4. Decreasing local node utilities (DLN). The minimum item utilities of descendant nodes for the node are decreased during the construction of a local UP-Tree.

Rationale: Assume that there is a list of promising items $<i_1', i_2', \ldots, i_n'>$ ordered by the descending order of local path utility values in a local header table. By the rationale of DGN strategy, the items $i_{k+1}', i_{k+2}', ..., i_n'$ and their utilities can be discarded from i_k'-*Tree*. Since the minimum item utility of an item i_m' $(k+1 \le m \le n)$ in a path is always equal to or less than its real utility in the path, we can also discard the minimum item utility of i_m' from the paths of i_k'-*Tree* without losing any PHUI.

The purpose of DLN strategy is similar to DGN strategy, while DLN strategy eliminates the minimum item utility values of descendants for a node in a local UP-Tree. It is applied during the insertion of the reorganized paths. When a reorganized path $p_j' = \{i_1, i_2, \ldots, i_n\}$ $(i_k \in I,\ 1 \le k \le n)$ is inserted into the conditional UP-Tree, the function ***Insert_Reorgnized_Path*** (R', i_1) is called, where R' is the root for the conditional UP-Tree.

The function ***Insert_Reorgnized_Path*** (N, i_x) takes a node N and an item i_x $(1 \le x \le n)$ as the inputs, which is performed as follows:

Line 1: If N has a child node S such that $S.item = i_x$, then increase $S.count$ by $p_j.count$; otherwise, a new child node S is created with $S.item = i_x$, $S.count = p_j'.count$ $S.parent = N$ and $S.nu = 0$.

Line 2: Increase $S.nu$ by $(pu(p_j', \{a_i\}\text{-}CPB) - p_j'.count \times \sum_{p=x+1}^{n} miu(i_p))$, where $i_p \in p_j'$ and $1 \le p \le n$.

Line 3: Call ***Insert_Reorgnized_Path*** $(S, i_{(p+1)})$ if $p \neq n$.

Example 8. Consider $\{D\}$'s conditional pattern base shown in Table 8, the reorganized transactions are shown in the second column, and their path utilities which are reduced by strategies DGU, DGN and DLU are shown in the last column. When the first reorganized path $\{C\}$ is inserted into the {*D*}-*Tree*, the first node $\{C\}$ is created under the root R' with $\{C\}.count = 1$ and $\{C\}.nu = 3$. When the second path $\{C, B, E\}$ is inserted into the tree, $\{C\}.count$ is increased by 1, and $\{C\}.nu$ is increased by $(20 - (miu(\{B\}) \times 1 + miu(\{E\}) \times 1)) = 20 - (4+3) = 13$. After that, $\{C\}.nu$ is equal to 16. The second node $\{B\}$ is crated under the node $\{C\}$ with $\{B\}.count = 1$ and $\{B\}.nu = (20 - miu(\{E\}) \times 1) = 20 - 3 = 17$. The last node $\{E\}$ is created under the node $\{B\}$ with $\{E\}.count = 1$ and $\{E\}.nu = 20$. After inserting all paths in {*D*}-*CPB*, {*D*}-*Tree* is constructed completely. Figure 4(b) shows a conditional UP-Tree for item $\{D\}$ when the four strategies are applied. Comparing with {*D*}-*Tree* shown in Figure 4(a), the node utilities of the nodes in {*D*}-*Tree* are further reduced by applying both strategies DLU and DLN. Therefore, the generation of PHUIs can be more efficient since fewer PHUIs are generated by applying the above four strategies. The proposed algorithm UP-Growth is developed based on the strategies DLU and DLN. The complete set of PHUIs is generated by recursively calling 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 the UP-Growth is shown as follows:

Subroutine: UP-Growth (T_x, H_x, X)
Input: A UP-Tree T_x, a header table H_x for T_x and an itemset X.
Output: All PHUIs in T_x.
Procedure UP-Growth (T_x, H_x, X)

(1) **For each** entry a_i in H_x **do**
(2) Generate a PHUI $Y = X \bigcup a_i$;
(3) The estimate utility of Y is set as a_i's utility value in H_x;
(4) Construct Y's conditional pattern base Y-*CPB*;
(5) Put local promising items in Y-*CPB* into H_y
(6) Apply strategy DLU to reduce path utilities of the paths;
(7) Apply strategy DLN and insert paths into T_y;
(8) **If** $T_y \neq null$ **then** call **UP-Growth**(T_y, H_y, Y);
(9) **End for**

3.3 Efficiently Identify High Utility Itemsets

In this part, high utility itemsets are identified by checking the real utilities of the PHUIs in the database. The purpose of this part is equivalent to that of phase II in [2, 7]. However, in previous work [2, 7], two problems in this phase occur: (1) The number of HTWUIs is too huge, and (2) scanning the original database is very time-consuming.

In our framework, the estimated utilities of PHUIs are smaller than or equal to the TWUs of HTWUIs since they are effectively reduced by the proposed four strategies. Thus, the number of PHUIs is much smaller than that of HTWUIs. Therefore, in phase II, our approach is much more efficient than the previous methods [2, 7]. Although our approach generates fewer candidates in phase I, scanning original database is still time-consuming since the original database is large and contains lots unpromising items. In view of these, in our framework, high utility itemsets are identified by scanning the reorganized transactions. Since there is no unpromising item in the reorganized transactions, the 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.

Table 9. Dataset characteristics

Dataset	N	T	D
T10I6D100K	1,000	10	100,000
Chess	76	37	3,196
BMS-Web-View-1	497	2.5	59,602

4. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our algorithm and compare it with IHUP algorithm [2]. The experiments were performed on a 2.66 GHz Intel Core 2 Quad Processor with 2 gigabyte memory, and running on Windows XP. The algorithms are implemented in Java language. Both synthetic and real datasets are used to evaluate the performance of the algorithms. Synthetic datasets were generated from the data generator in [1]. The parameters are described as follows: *D* is the total number of transactions; *T* is the average size of transactions; *N* is the number of distinct items; *I* is the average size of maximal potential frequent itemsets. The utility table and the quantity of each item are generated as the settings in [7]. Real world datasets BMS-Web-View-1 and Chess were obtained from FIMI Repository [11]. Table 9 shows the characteristics of the datasets in the experiments.

For comparing the performance of the proposed algorithms, we design three compared algorithms and give them new notations as follows. The proposed algorithm, including UP-Tree (using DGU and DGN) and UP-Growth (using DLU and DLN), is denoted as UP+UPG. The algorithm proposed in [2] is denoted as IHUP+FPG, since it uses IHUP-Tree and FP-Growth. UP+UPG means the PHUIs are generated from UP-Tree by applying UP-Growth. IHUP+FPG means the HTWUIs are generated from IHUP-Tree by applying FP-Growth. To further compare the performance of FP-Growth and UP-Growth, an algorithm which is called UP+FPG is also proposed. Different from UP+UPG, UP+FPG generates PHUIs from UP-Tree by FP-Growth rather than UP-Growth. In UP+FPG, only DGU and DGN are applied in UP-Tree. In the above algorithms, both UP-Tree and IHUP-Tree are constructed by scanning database twice. The items in a transaction are rearranged in descending order of the global TWU during the construction of both UP-Tree and IHUP-Tree. In phase II, the three algorithms identify high utility itemsets by scanning the database which contains no unpromising items. For convenience, PHUIs and HTWUIs are both called the candidates for high utility itemsets in the following experiments.

4.1 Evaluation on Synthetic Datasets

We first show the performance of the algorithms on the synthetic dataset T10I6D100K. Table 10 shows the number of candidates on T10I6D100K under varied minimum utility thresholds. As shown in Table 10, UP+FPG generates fewer candidates than IHUP+FPG since the node utilities of the nodes in the UP-Tree are less than the IHUP-Tree. This shows the effectiveness of strategies DGU and DGN. By applying strategy DGU, global unpromising items and their utilities are discarded from the transactions and TUs. By applying DGN strategy, the node utilities of the nodes in a global UP-Tree are effectively decreased since the utilities of their descendants are discarded. In Table 10, when the minimum utility threshold is less than 0.6%, the number of candidates generated by UP+UPG becomes smaller than UP+FPG obviously. This indicates that strategies DLU and DLN work well and more itemsets which are impossible to be high utility are reduced than FP-Growth when the minimum utility threshold is low. By applying DLU strategy, local unpromising items are removed from the paths of conditional pattern base and their minimum item utilities are eliminated from the path utilities. By applying DLN, the node utilities of the nodes in a local UP-Tree are decreased since they discard the minimum item utilities of their descendants.

Table 10. Number of candidates on T10I6D100K

Minimum utility	IHUP+FPG	UP+FPG	UP+UPG
0.2%	20,651	15,057	10,664
0.4%	4,003	2,347	1,990
0.6%	1,684	910	844
0.8%	873	527	521
1.0%	566	411	411

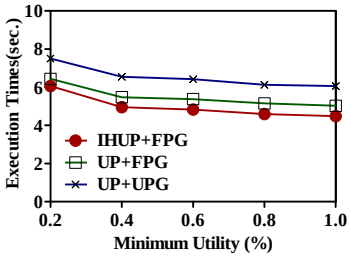


Figure 5. Execution time for phase I on T10I6D100K.

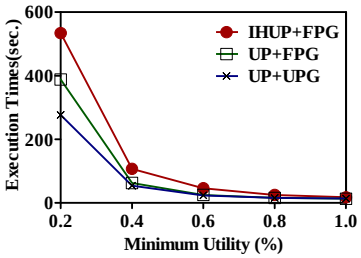


Figure 6. Execution time for phase II on T10I6D100K.

Figure 5 and Figure 6 show the execution time for phase I and phase II, respectively. In Figure 5, UP+UPG and UP+FPG spend slightly more execution time, i.e., about one second, than IHUP+FPG. It is because the former two methods perform additional processes to decrease the overestimated utilities of the itemsets. Thus, as shown in Figure 6, the execution time of UP+UPG and UP+FPG in phase II is much less than IHUP+FPG by around two hundred seconds. It shows that it is worth spending a little more execution time for decreasing the overestimated utilities for pruning candidates. By the above observation, we show that the overall performance of UP+UPG and UP+FPG outperforms IHUP+FPG. The results are reasonable since the more candidates are produced in phase I, the more candidates need to be identified in phase II.

4.2 Evaluation on Real Datasets

In this section, we compare the performance of the algorithms on real datasets. We first show the evaluation on Chess dataset [11], which is a well-known dense dataset in which each transaction contains 37 items. Table 11 shows the number of candidates on Chess dataset under different minimum utility threshold. As shown in Table 11, more than forty thousand candidates are

Table 11. Number of candidates on Chess

Minimum utility	IHUP+FPG	UP+FPG	UP+UPG
30%	38,686,975	4,108,608	1,592,414
40%	6,682,935	79,766	3,527
50%	1,320,445	37	37
60%	264,418	34	34
70%	49,039	24	24

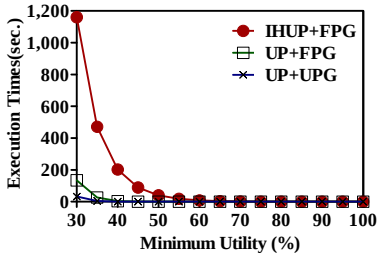


Figure 7. Execution time for phase I on Chess.

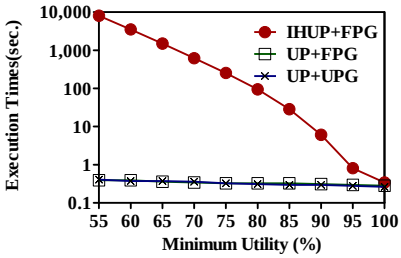


Figure 8. Execution time for phase II on Chess.

Table 12. Number of candidates on BMS-Web-View-1

Minimum utility	IHUP+FPG	UP+FPG	UP+UPG
2.9%	29,561,924	206	206
3.0%	566,651	164	164
3.2%	2,010	133	133
3.6%	387	108	108
4.0%	170	76	76

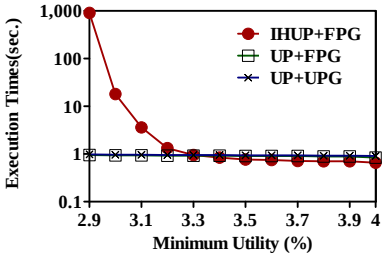


Figure 9. Execution time for phase I on BMS-Web-View-1.

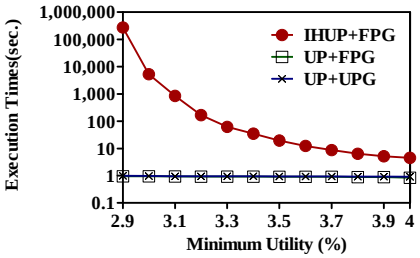


Figure 10. Execution time for phase II on BMS-Web-View-1.

generated by IHUP+FPG when the minimum utility threshold is 70%. However, the number of candidates generated by UP+UPG and UP+FPG is just 24 under the same threshold. On average, the number of candidates generated by UP+UPG and UP+FPG is more than 2,000 times smaller than the number of candidates generated by IHUP+FPG. The reason is that Chess dataset contains lots of long transactions. When a transaction which contains lots of items is inserted into the IHUP-Tree, the node utilities of all related nodes are increased by the large transaction utility value. However, by applying strategies DGN and DLN, the more promising items are in a transaction, the more node utilities can be reduced in UP-Tree. Therefore, UP+UPG and UP+FPG produce much less candidates than IHUP+FPG.

Figure 7 and Figure 8 show the execution time for phase I and phase II on Chess dataset, respectively. In Figure 7, the execution time of UP+UPG is about 1,000 times faster than IHUP+FPG when the minimum utility threshold is around 30%. This is because that IHUP+FPG recursively constructs conditional IHUP-Tree by FP-Growth and it takes lots of time to generate the candidates. In Figure 8, the performance of IHUP+FPG becomes much worse than both UP+FPG and UP+UPG when the minimum utility threshold is less than 50%. This is because that IHUP+FPG generates large amount of candidates in phase I and there are too many PHUIs need to be identified in phase II. On the contrary, UP+FPG and UP+UPG only spend one second in phase II since the number of candidates generated by the two approaches is quite small. Table 11, Figure 7 and Figure 8 show that UP-Tree and UP-Growth have better performance than IHUP-Tree and FP-Growth even in dense datasets.

Table 12 shows the number of candidates on BMS-Web-View- 1 dataset when the minimum utility threshold is between 2.9% and 4%. When the threshold is higher than 4%, the number of candidates of the three algorithms is the same. In Table 12, the number of candidates generated by IHUP+FPG is about 100,000 times larger than that generated by UP+UPG and UP+FPG when the threshold is 2.9%. The reason is that BMS-Web-View-1 dataset contains some long transactions which contain more than one hundred items and the utility of the itemset is highly overestimated by the IHUP+FPG. It causes IHUP+FPG generates a large amount of candidates in phase I on this dataset.

Figure 9 and Figure 10 show the execution time of the three algorithms in phase I and phase II on BMS-Web-View-1, respectively. In Figure 9, IHUP+FPG performs a little better than UP+UPG and UP+FPG when the minimum utility threshold is above than 3.3% since the construction of UP-Tree requires more processing time than IHUP-Tree. Although IHUP+FPG is slightly faster than UP+UPG and UP+FPG when the threshold is above than 3.3%, the three algorithms only spend about 1 second to generate candidates. When the threshold is less than 3.3%, the execution time of IHUP+FPG increases sharply. When the threshold is less than 2.9%, UP+UPG and UP+FPG outperform IHUP+FPG over than 1,000 times in phase I. In Figure 10, the execution time of IHUP+FPG increases sharply with decreasing minimum utility threshold. The reason is that there are too many

Table 13. Number of candidates on different data size

Database size	IHUP + FPG	UP + FPG	UP + UPG	Number of high utility itemsets
200K	70,751	60,042	47,867	8,158
400K	74,666	64,658	52,340	8,980
600K	68,991	59,941	48,103	7,323
800K	68,083	59,513	48,159	8,103
1,000K	68,983	58,279	46,754	7,142

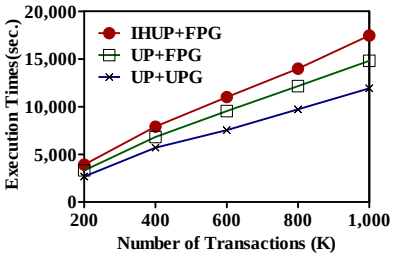


Figure 11. Scalability for the algorithms.

candidates needed to be identified for IHUP+FPG. In other words, the search space of IHUP+FPG in phase II is very large. When the threshold is less than 3%, the execution time of UP+UPG and UP+FPG is over 10,000 times better than IHUP+FPG.

4.3 Scalability

In the following experiments, we vary the dataset size of T10I6 dataset to evaluate the scalability for the three algorithms. Table 13 shows the number of candidates generated by these algorithms. The last column in Table 13 shows the number of high utility itemsets which is identified after phase II. The results of the three algorithms are identical. Figure 11 shows the total execution time for the three algorithms when the minimum utility threshold is set as 0.1%. As shown in Figure 11, all the three algorithms have a good scalability. However, the execution time of UP+UPG and UP+FPG is less than IHUP+FPG since IHUP+FPG generate more candidates than the two approaches. When the size of database increases, the execution time for identifying each high utility itemset also increases. Therefore, IHUP+FPG requires more processing time than UP+UPG and UP+FPG.

The experimental results show that our approach outperforms the state-of-the-art algorithms on both real and synthetic datasets. The main reasons for the improvements are summarized as follows. First, the information about high utility itemsets are maintained in the global UP-Tree, where the node utilities of the nodes are much less than the TWUs of the nodes in IHUP-Tree since strategies DGU and DGN are applied during the construction of a global UP-Tree. Second, UP-Growth generates much fewer candidates than FP-Growth since strategies DLU and DLN are applied during the construction of a local UP-Tree. By the proposed algorithm and strategies, generation of candidates can be more efficient in phase I since lots of useless candidates are pruned. Third, the high utility itemsets are efficiently identified from the set of PHUIs by scanning the database without containing unpromising items. Therefore, our approach achieves a better performance than IHUP algorithm.

5. CONCLUSIONS

In this paper, we have proposed an efficient algorithm named UP-Growth for mining high utility itemsets from transaction databases. A data structure named UP-Tree is proposed for maintaining the information of high utility itemsets. Hence, the potential high utility itemsets can be efficiently generated from the UP-Tree with only two scans of the database. Besides, we develop four strategies to decrease the estimated utility value and enhance the mining performance in utility mining. In the experiments, both of synthetic and real datasets are used to evaluate the performance of our algorithm. The mining performance is enhanced significantly since both the search space and the number of candidates are effectively reduced by the proposed strategies. The experimental results show that UP-Growth outperforms the state-of-the-art algorithms substantially, especially when the database contains lots of long transactions.

6. ACKNOWLEDGMENTS

This research was supported by National Science Council, Taiwan, R.O.C. under grant no. NSC 96-2221-E-006-143-MY3 and in part by NSF, USA through grants IIS-0905215 and IIS-0914934.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, pp. 487-499, 1994.

[2] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee. Efficient tree structures for high utility pattern mining in incremental databases. In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, Issue 12, pp. 1708-1721, 2009.

[3] R. Chan, Q. Yang, and Y. Shen. Mining high utility itemsets. In *Proc. of Third IEEE Int'l Conf. on Data Mining*, pp. 19-26, Nov., 2003.

[4] A. Erwin, R. P. Gopalan, and N. R. Achuthan. Efficient mining of high utility itemsets from large datasets. In *Proc. of PAKDD 2008, LNAI 5012,* pp. 554-561.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the ACM-SIGMOD Int'l Conf. on Management of Data*, pp. 1-12, 2000.

[6] Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Isolated items discarding strategy for discovering high utility itemsets. In *Data & Knowledge Engineering*, Vol. 64, Issue 1, pp. 198-217, Jan., 2008.

[7] Y. Liu, W. Liao, and A. Choudhary. A fast high utility itemsets mining algorithm. In *Proc. of the Utility-Based Data Mining Workshop*, 2005.

[8] B.-E. Shie, V. S. Tseng, and P. S. Yu. Online mining of temporal maximal utility itemsets from data streams. In *Proc. of the 25th Annual ACM Symposium on Applied Computing*, Switzerland, Mar., 2010.

[9] H. Yao, H. J. Hamilton, L. Geng, A unified framework for utility-based measures for mining itemsets. In *Proc. of ACM SIGKDD 2nd Workshop on Utility-Based Data Mining*, pp. 28-37, USA, Aug., 2006.

[10] S.-J. Yen and Y.-S. Lee. Mining high utility quantitative association rules. In *Proc. of 9th Int'l Conf. on Data Warehousing and Knowledge Discovery, Lecture Notes in Computer Science 4654*, pp. 283-292, Sep., 2007.

[11] Frequent itemset mining implementations repository, http://fimi.cs.helsinki.fi/