Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments

Bai-En Shie[1], Hui-Fang Hsiao[1], Vincent S. Tseng[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
{brianshie, karolter1130}@gmail.com, tsengsm@mail.ncku.edu.tw, psyu@cs.uic.edu

Abstract. Mining user behaviors in mobile environments is an emerging and important topic in data mining fields. Previous researches have combined moving paths and purchase transactions to find mobile sequential patterns. However, these patterns cannot reflect actual profits of items in transaction databases. In this work, we explore a new problem of mining high utility mobile sequential patterns by integrating mobile data mining with utility mining. To the best of our knowledge, this is the first work that combines mobility patterns with high utility patterns to find high utility mobile sequential patterns, which are mobile sequential patterns with their utilities. Two tree-based methods are proposed for mining high utility mobile sequential patterns. A series of analyses on the performance of the two algorithms are conducted through experimental evaluations. The results show that the proposed algorithms deliver better performance than the state-of-the-art one under various conditions.

Keywords: High utility mobile sequential pattern; utility mining; mobility pattern mining; mobile environment

1 Introduction

With the rapid development of tele-communication technologies, mobile devices and wireless applications become increasingly popular. One's current position can be acquired via a mobile device with GPS service. With a series of users' moving logs, we can know the moving paths of mobile users. Besides, a greater number of people are using mobile devices to purchase mobile services online by credit cards. Combining moving logs and payment records, *mobile transaction sequences*, which are the sequences of moving paths with transactions, are obtained. Yun et al. [14] first proposed a framework for discovering *mobile sequential patterns*, i.e., the sequential patterns with their moving paths in mobile transaction sequence databases. Mobile sequential patterns can be applied in many applications, such as route planning in mobile commerce environment and maintaining website structures of online shopping websites.

However, in mobile sequential pattern mining, the importance of items is not considered. In the framework of traditional frequent pattern mining, utility mining [3, 4, 7, 8, 12, 13] is proposed for solving this problem. Instead of finding frequent patterns,

utility mining discovers the patterns with high utilities, which are called *high utility patterns*. By utility mining, patterns with higher importance/profit/user interests can be found. For instance, the frequent patterns involving refrigerators may not be easily found from the transaction databases of hypermarkets since the frequency of purchasing refrigerators is much less than that of other items. But if we apply utility mining, the high utility patterns about refrigerators may be found since the utilities, i.e., the profits, of refrigerators are higher than that of others. Therefore, it is obvious that pushing utility mining into the framework of mobility pattern mining is an essential topic. If decision makers know which patterns are more valuable, they can choose more appropriate actions based on the useful information. Considering the utilities of items in customers' frequent purchasing patterns and moving paths is crucial in many domains, such as finding valuable patterns in mobile commerce environments, metropolitan planning and maintaining the structure and designing promotions for online shopping websites.

In view of the above issues, we aim at integrating mobility pattern mining with utility mining to find *high utility mobile sequential patterns* in this research. The proposed pattern must be not only high utility but also frequent. In other words, it is composed of both high utility purchasing pattern and frequent moving path. This is because applying only utility mining to the mobile environments is insufficient. A moving path with high utility but low frequency is unpractical. Users may be confused with a number of these redundant patterns. By this consideration, the proposed pattern is more useful than the patterns that apply only utility mining or frequent pattern mining to the mobile environments.

In this paper, we propose two tree-based methods, namely $UMSP_{DFG}$ (*mining high Utility Mobile Sequential Patterns with a tree-based Depth First Generation strategy*) and $UMSP_{BFG}$ (*mining high Utility Mobile Sequential Patterns with a tree-based Breadth First Generation strategy*). The main difference of the two algorithms is the method for generating the length 2 patterns during the mining process, which is the bottleneck of pattern mining. Both of the algorithms use a tree structure *MTS-Tree* (*Mobile Transaction Sequence Tree*) to summarize the information about locations, items, paths and utilities in mobile transaction databases. To the best of our knowledge, this is the first work that explores the integration of mobility pattern mining and utility mining. The experimental results show that $UMSP_{BFG}$ has better performance than $UMSP_{DFG}$. Moreover, the performance of the two proposed tree-based methods outperforms the compared level-wise algorithm which is improved by the state-of-the-art mobile sequential pattern algorithm [14].

Major contributions of this work are described as follows. First, this research is the first work that integrates high utility pattern mining with mobility pattern mining so as to explore the new problem of mining high utility mobile sequential patterns. Second, different methods proposed under different pattern generation strategies are proposed for solving this problem. Third, a series of detailed experiments is conducted to evaluate the performance of the proposed methods in different conditions. By the combination of high utility patterns and moving paths, highly profitable mobile sequential patterns can be found. We expect that the useful patterns can bring novel and insightful information in mobile commerce environments.

The remainder of this paper is organized as follows. We briefly review the related work in section 2. Section 3 is the problem definition of this research. In section 4, we describe the proposed algorithms. The experimental evaluation for performance study is made in section 5. The conclusions and future work are given in section 6.

2 Related Work

Extensive studies have been proposed for finding frequent patterns in transaction databases [1, 2, 5, 10]. Frequent itemset mining [1, 5] is the most popular topic among them. Apriori [1] is the pioneer for mining frequent itemsets from transaction databases by a level-wise candidate generation-and-test method. Tree-based frequent itemset mining algorithms such as FP-Growth [5] were proposed afterward. FP-Growth improves the efficiency of frequent itemset mining since it does not have to generate candidate itemsets during the mining process and it only scans the database twice. Afterwards, sequential pattern mining [2, 10] is proposed for finding customer behaviors in transaction databases. As an extension method of Apriori, AprioriAll [2] also use a level-wise framework to find sequential patterns. On the contrary, PrefixSpan [10] finds sequential patterns directly from projected databases without generating any candidate pattern. Thus, the performance can be more improved.

Mining user behaviors in mobile environments [6, 9, 11, 14] is an emerging topic in the frequent pattern mining field. SMAP-Mine [11] was first proposed for finding customers' mobile access patterns. However, in different time periods, users' popular services may be totally different. Thus, T-Map algorithm [6] was proposed to find temporal mobile access patterns in different time intervals. Although users' mobile access patterns are important, their moving paths are also essential. Therefore, Yun et al. [14] proposed a framework which combines moving paths and sequential patterns to find mobile sequential patterns. Moreover, Lu et al. [9] proposed a framework for discovering cluster-based mobile sequential patterns. The customers whose moving paths and transactions are similar will be grouped into the same clusters. By this framework, the discovered patterns may be closer to the customer behaviors in real life.

In the above researches, the profits of items are not considered yet. In transaction databases, items have different profits. Utility mining [3, 4, 7, 8, 12, 13] is proposed to conquer this problem. Among these researches, Liu et al. [8] proposed Two-Phase algorithm which utilizes the transaction-weighted downward closure property to maintain the downward closure property in the processes of utility mining. On the other hand, Ahmed et al. [3] employed a tree structure, named IHUP-Tree, to maintain essential information about utility mining. Different from Two-Phase, it avoids scanning database multiple times and generating candidate patterns. Although IHUP-Tree achieves a better performance than Two-Phase, it still produces too many high transaction weighted utilization itemsets. Therefore, Tseng et al. proposed UP-Growth [12], which applies four strategies for decreasing the estimated utilities during the mining processes. By these strategies, the number of possible high utility itemsets is effectively reduced and the performance of utility mining is further improved.

By the above literature reviews, although there are many researches about mobility pattern mining and utility mining, there is no research about the combination of the two topics. This paper is the first work which integrates the two topics to find high utility patterns with frequent moving paths in mobile environments.

3 Problem Definition

In this section, we define basic notations for mining high utility mobile sequential patterns in mobile environments in detail.

Table 1. Mobile transaction sequence database *DB*

SID	Mobile transaction sequence	SU
S_1	<(A; {[i_1, 2]}), (B; null), (C; {[i_2, 1]}), (D; {[i_4, 1]}), (E; null), (F; {[i_5, 2]})>	54
S_2	<(A; {[i_1, 3]}), (B; null), (C; {[i_2, 2], [i_3, 5]}), (K; null), (E; {[i_6, 10]}), (F; {[i_5, 4]}), (G; {[i_8, 2]}), (L; null), (H; {[i_7, 2]})>	132
S_3	<(A; {[i_1, 3]}), (B; null), (C; {[i_2, 1], [i_3, 5]}), (D; {[i_4, 2]}), (E; null), (F; {[i_5, 1], [i_6, 2]}), (G; null), (H; {[i_7, 1]})>	72
S_4	<(A; {[i_1, 1]}), (W; null), (C; {[i_3, 10]}), (E; null), (F; {[i_5, 1]}), (G; {[i_8, 2]}), (L; null), (H; {[i_7, 1]}), (E; {[i_9, 1]})>	59
S_5	<(A; {[i_1, 4]}), (B; null), (C; {[i_3, 10]}), (D; {[i_4, 1]}), (E; null), (F; {[i_5, 1]}), (G; null), (H; {[i_7, 2]})>	73
S_6	<(C; {[i_2, 2]}), (D; null), (E; {[i_9, 1]}), (F; {[i_5, 1]})>	31

Table 2. Utility table

Item	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9
Utility	1	5	3	11	18	2	5	1	3

Let $L = \{l_1, l_2, \ldots, l_p\}$ be a set of *locations* in a mobile commerce environment and $I = \{i_1, i_2, \ldots, i_g\}$ be a set of *items* sold at the locations. An *itemset* is denoted as $\{i_1, i_2, \ldots, i_k\}$, where each item $i_v \in I$, $1 \le v \le k$ and $1 \le k \le g$. Given a *mobile transaction sequence database D*, a *mobile transaction sequence* $S = <T_1, T_2, \ldots, T_n>$ is a set of transactions ordered by time, where a *transaction* $T_j = (l_j; \{[i_{j_1}, q_{j_1}], [i_{j_2}, q_{j_2}], \ldots, [i_{j_h}, q_{j_h}]\})$ represents that a user made T_j in l_j, where $1 \le j \le n$. In T_j, the *purchased quantity* of item i_{j_p} is q_{j_p}, where $1 \le p \le h$. A *path* is denoted as $l_1\, l_2 \ldots l_r$, where $l_j \subset L$ and $1 \le j \le r$.

Definition 1: A *loc-itemset* $<l_{loc}; \{i_1, i_2, \ldots, i_g\}>$ signifies the itemset $\{i_1, i_2, \ldots, i_g\}$ was traded in l_{loc}, where $l_{loc} \in L$ and $\{i_1, i_2, \ldots, i_g\} \subseteq I$. The utility of a loc-itemset $Y = <l_{loc}; \{i_1, i_2, \ldots, i_g\}>$ in a mobile transaction sequence database D is denoted as $u(Y)$ and defined as $\sum_{S_j:(Y \subseteq S_j) \wedge (S_j \in D)} \sum_{k=1}^{g} u(<l_{loc}; i_k>, S_j)$, where $u(<l_{loc}; i_k>, S_j)$, defined as $w(i_k) \times q_{j_k}$, is the utility of the loc-item $<l_{loc}; i_k>$ in the mobile transaction sequence S_j. $w(i_k)$ is the *unit profit* of the item i_k recorded in a *utility table*.

Take the mobile transaction sequence database in Table 1 and the utility table in Table 2 as an example, $u(<C; \{i_2, i_3\}>) = u(<C; \{i_2, i_3\}>, S_2) + u(<C; \{i_2, i_3\}>, S_3) = (5 \times 2 + 3 \times 5) + (5 \times 1 + 3 \times 5) = 25 + 20 = 45$.

Definition 2: A *loc-pattern* X is a list of loc-itemsets. It is denoted as $<l_1; \{i_{1_1}, i_{2_1}, \ldots, i_{g_1}\}><l_2; \{i_{1_2}, i_{2_2}, \ldots, i_{g_2}\}> \ldots <l_m; \{i_{1_m}, i_{2_m}, \ldots, i_{g_m}\}>$. The utility of a loc-pattern X in S_j is denoted as $u(X, S_j)$ and defined as $\sum_{\forall Y \in X} u(Y, S_j)$. The utility of a loc-pattern X in D is denoted as $u(X)$ and defined as $\sum_{(X \subseteq S_j) \wedge (S_j \in D)} u(X, S_j)$.

For instance, for the loc-pattern $X_1 = <A; i_1><C; \{i_2, i_3\}>$ in Table 1, $u(X_1, S_2) = u(<A; i_1>, S_2) + u(<C; \{i_2, i_3\}>, S_2) = 28$, and $u(X_1) = u(X_1, S_2) + u(X_1, S_3) = 28 + 23 = 51$.

Definition 3: A *moving pattern* is composed of a loc-pattern and a path. The utility of a moving pattern $P = <\{<l_1; \{i_{1_1}, i_{2_1}, \ldots, i_{g_1}\}><l_2; \{i_{1_2}, i_{2_2}, \ldots, i_{g_2}\}> \ldots <l_m; \{i_{1_m}, i_{2_m}, \ldots, i_{g_m}\}>\};$

$l_1 l_2 ... l_m$>, denoted as $u(P)$, is defined as the sum of utilities of the loc-patterns in P in the mobile transaction sequences which contain the path of P in D. The *support* of a moving pattern P, denoted as $sup(P)$, is defined as the number of mobile transaction sequences which contain P in D. Similarly, the support of a loc-itemset or a loc-pattern is also defined as the number of mobile transaction sequences which contain it in D.

For example, for the moving path P_1 = <{<A; i_1><C; {i_2, i_3}>}; ABC>, $u(P_1)$ = u(<A; i_1><C; {i_2, i_3}>, S_2) + u(<A; i_1><C; {i_2, i_3}>, S_3) = 28+23 = 51, and $sup(P_1)$ = 2.

Definition 4: Given a *minimum support threshold* δ and a *minimum utility threshold* ε, a moving pattern P is called a *high utility mobile sequential pattern*, abbreviated as *UMSP*, if $sup(P) \geq \delta$ and $u(P) \geq \varepsilon$. The length of a pattern is the number of loc-itemsets in this pattern. A pattern with length k is denoted as k-pattern.

For example, in Table 1, if δ = 2 and ε = 50, the moving pattern P_1 = <{<A; i_1><C; {i_2, i_3}>}; ABC> is a 2-UMSP since $sup(P_1) \geq 2$ and $u(P_1) > 50$.

After addressing the problem definition of mining high utility mobile sequential patterns in mobile environments, we introduce the *sequence weighted utilization* and *sequence weighted downward closure* property (abbreviated as SWDC), which are extended from [8].

Definition 5: The *sequence utility* of mobile transaction sequence S_j is denoted as $SU(S_j)$ and defined as the sum of the utilities of all items in S_j.

For example, $SU(S_6) = u$(<C; i_2>, S_6)+u(<E; i_9>, S_6)+u(<F; i_5>, S_6) = 10+3+18 = 31.

Definition 6: The *sequence weighted utilization*, abbreviated as SWU, of a loc-itemset, a loc-pattern, or a moving pattern is defined as the sum of SU of the mobile transaction sequences which contain it in D.

For example, SWU(<D; i_4>) = $SU(S_1)$+$SU(S_3)$+$SU(S_5)$ = 54+72+73 = 199; SWU(<A; i_1><C;{i_2,i_3}>) = $SU(S_2)$+$SU(S_3)$ = 132+72 = 204; SWU(<{<A;i_1><C;{i_2,i_3}>}; ABC>) = SWU(<A; i_1><C;{i_2, i_3}>, S_2)+SWU(<A; i_1><C;{i_2, i_3}>, S_3) = 132+72 = 204.

Definition 7: A pattern Y is called a *high sequence weighted utilization pattern*, if $sup(Y) \geq \delta$ and $SWU(Y) \geq \varepsilon$. In the following paragraphs, *high sequence weighted utilization loc-itemset*, *high sequence weighted utilization loc-pattern* and *high sequence weighted utilization mobile sequential pattern* are abbreviated as WULI, WULP and WUMSP, respectively.

Property 1. (Sequence weighted downward closure property): For any pattern P, if P is not a WUMSP, any superset of P is not a WUMSP.

Proof: Assume that there is a pattern P and P' is a superset of P. By Definition 6, $SWU(P) \geq SWU(P')$. If $SWU(P) < \varepsilon$, $SWU(P') < \varepsilon$. Similarly, by Definition 3, $sup(P) \geq sup(P')$. If $sup(P) < \delta$, $sup(P') < \delta$. By the above two conditions, we can obviously know that if P is not a WUMSP, any superset of P is not a WUMSP. ■

Problem Statement. Given a mobile transaction sequence database, a pre-defined utility table, a minimum utility threshold and a minimum support threshold, the problem of mining high utility mobile sequential patterns from the mobile transaction sequence database is to discover all high utility mobile sequential patterns whose supports and utilities are larger than or equal to the two thresholds in this database.

4 Proposed Methods

4.1 Algorithm $UMSP_{DFG}$

The workflow of the proposed algorithm $UMSP_{DFG}$ (*high Utility Mobile Sequential Pattern mining with a tree-based Depth First Generation strategy*) is shown in Figure 1. In step 1, WULIs and a mapping table are generated. Then a *MTS-Tree* (*Mobile Transaction Sequence Tree*) is constructed in step 2. In step 3, WUMSPs are generated by mining the MTS-Tree with the depth first generation strategy. Finally in step 4, UMSPs are generated by checking the actual utility of WUMSPs. In this section, we describe the construction of MTS-tree first and then address the generation of WUMSP.

We first address the process of generating WULIs by an example. Take the mobile transaction sequence database in Table 1 and the utility table in Table 2 for example. Assume the minimum support threshold is 2 and the minimum utility threshold is 100. In the first step, WULIs whose supports and SWUs are larger than or equal to the two thresholds are generated by the processes similar to [8]. In this case, eight WULIs shown in Table 3 are generated. Note that they are also 1-WULPs. Then the 1-WULPs are mapped sequentially into a mapping table as shown in Table 3.

4.1.1 The construction of MTS-Tree

The procedures of MTS-Tree construction are shown in Figure 2. The construction of MTS-Tree is completed after one scan of the original database. Without loss of generality, we give a formal definition for MTS-Tree first.

Definition 8. (MTS-Tree): In MTS-Tree, each node N includes $N._{location}$, $N._{itemset}$, $N._{SID}$ and a path table. N is represented by the form $<N._{location}$ $[N._{itemset1}]$: $N._{SID1}$; $[N._{itemset2}]$: $N._{SID2}$; ... >. $N._{location}$ records the node's location. Each node has several itemsets $N._{itemset}$ which represent the itemsets traded in the same location. For each itemset in a node, it has a string of sequence identifiers, $N._{SID}$, which records the mobile transaction sequences with the item in it. A *path table* records the paths, which are a series of locations with no item purchased from N's parent node to N, and the SIDs of the paths. Moreover, a *header table* is applied to efficiently traverse the nodes of a MTS-Tree. In a header table, each *entry* is composed of a 1-WULP, its SWU and support, and a link which points to its first occurrence in MTS-Tree.

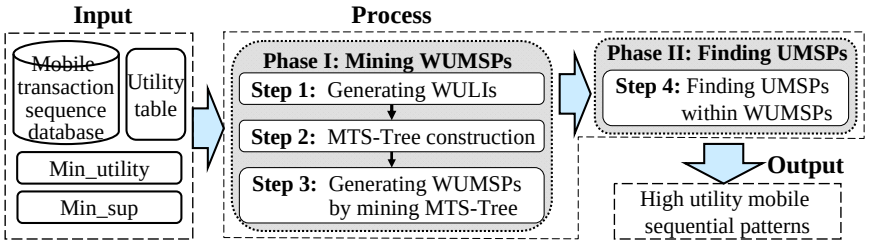


Fig. 1. The framework of the proposed algorithm $UMSP_{DFG}$.

Table 3. Mapping table

1-WULP	A;i_1	C;i_2	C;i_3	D;i_4	F;i_5	G;i_8	H;i_7	C;{i_2,i_3}
After Mapping	A;t_1	C;t_2	C;t_3	D;t_4	F;t_5	G;t_6	H;t_7	C;t_8

```
Algorithm (Step 2 of UMSP_DFG: MTS-Tree construction)
Input: Mobile transaction sequence database DB, mapping table MT
Output: MTS-Tree
1. create a header table H
2. create a root R for an MTS-Tree T
3. foreach mobile transaction sequence S_i in DB do
4.   let path_start = false
5.   call InsertMTS_Tree(S_i, R, MT, sid)

Procedure InsertMTS_Tree(S_i, R, MT, sid)
1. if S_i is not NULL then
2.   divide S_i into [x|X]
    /* x: the first loc-itemset of S_i. X: the remaining list of S_i */
3.   let temppath = NULL
4.   if there is a combination y' of x exists in MT then
5.     convert x to the HTWULI y in MT
6.     if R has a child node C where C.location = y.location then
7.       if y.item exists in C.items then
8.         insert sid into C.[y.item].sid
9.       else
10.        create a new item y.item to C.items
11.        insert sid to C.[y.item].sid
12.    else
13.      create a new node C as a child node of R
14.      let C.location = y.location
15.      let C.item = y.item
16.      append sid to C.[y.item].sid
17.    update y's WULI, sup, TWU in H
18.    if temppath = NULL then
19.      append temppath and sid to C.pathtable
20.      let temppath = NULL
21.  else
22.    append x.location to temppath
23.  if X is not NULL then
24.    call InsertMTS_Tree(X, C, MT, sid)
```

Fig. 2. The procedure of MTS-Tree construction

Now we introduce the processes of the MTS-Tree construction, i.e., the second step of $UMSP_{DFG}$, by continuing the example in Section 4.1. At the beginning, the first mobile transaction sequence S_1 is read. The first transaction in S_1 is (A; {[i_1, 2]}), so we check the loc-itemset <A; i_1> in the mapping table in Table 3. After checking, the loc-itemset is converted into <A; t_1>, it is then inserted into MTS-Tree. Since there is no corresponding node in the MTS-Tree, a new node <A[t_1]: S_1> is created. The loc-itemset <A; t_1> is also inserted as an entry into the header table.

Subsequently, the second transaction (B; null) is evaluated. Since it has no purchased item, the location B is kept as a temporary path. Then the third transaction (C; {[i_2, 1]}) is checked in the mapping table and converted into <C; t_2>. Then the node <C[t_2]: S_1> is created and inserted as a child node of <A[t_1]: S_1>. Since there is a path B in the temp path, B and its SID S_1 are recorded into the path table of the node

<C[t_2]: S_1>. Then the information in the temp path is cleared. The loc-itemset <C; t_2> and its relevant information are also inserted into the header table. The remaining transactions in S_1 are inserted into the MTS-Tree sequentially by the same way.

Subsequently, the second mobile transaction sequence S_2 is read. For the first transaction (A; {[i_1, 3]}), it is converted into <A; t_1>. Since there is already a node <A[t_1]: S_1> with the same location A in MTS-Tree, the SID S_2, SWU and support of the transaction are updated in the node and its entry in the header table, respectively. Then the location B of the second transaction (B; null) is kept into the temp path. Next, the third transaction (C; {[i_2, 2], [i_3, 5]}) is evaluated. By the mapping table, it is first converted into <C; t_2>, <C; t_3> and <C; t_8>. Since there is a node <C[t_2]: S_1> with location C and item t_2, <C; t_2> can be just updated on the SID in the node by the processes mentioned above. On the other hand, for <C; t_3> and <C; t_8>, their items and SIDs are stored into the node. After processing this transaction, the node becomes <C [t_2]: S_1 S_2; [t_3]: S_2; [t_8]: S_2>. The remaining transactions in S_2 are inserted into the MTS-Tree sequentially by the same way. After all sequences in *D* are inserted, we can get the MTS-Tree in Figure 3.

4.1.2 Generating WUMSPs from MTS-Tree

After constructing MTS-Tree, now we show the step 3 of $UMSP_{DFG}$. The purpose of this step is generating WUMSPs from MTS-Tree by the *depth first generation* strategy. The procedures are shown in Figure 4. First, WULPs and their *conditional MTS-Trees* are generated by tracing the links of the entries in the header table of the MTS-Tree. Then the WULPs are inserted into a *WUMSP-Tree* (*high sequence Weighted Utilization Mobile Sequential Pattern Tree*), which is used for storing the WUMSPs. Then the paths of the WULPs in the WUMSP-Tree are traced in the original MTS-Tree and the WUMSPs are generated.

Definition 9. (WUMSP-Tree): In a WUMSP-Tree, each node is a WULI. For a node *N*, $N._{SID}$ and its path table are recorded. *N* is represented by the form <$N._{WULI}$: $N._{SID}$>. For the WULI in a node, it has a string of sequence identifiers, $N._{SID}$, which records the mobile ransaction sequences with the WULI occurring in it. A path table records the paths from the node *N* to root and the SIDs of the paths.

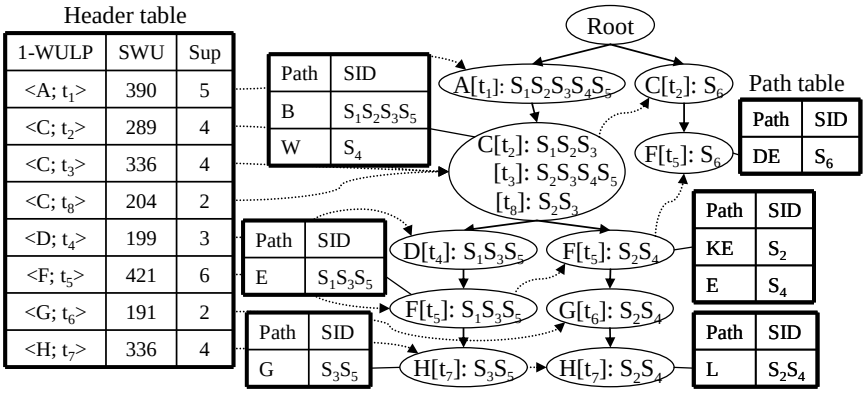
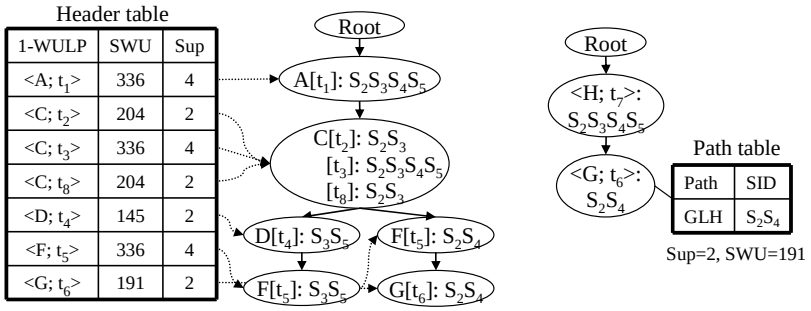


Fig. 3. An Example of MTS-Tree.

```
Algorithm (Step 3 of UMSP_DFG: Generating WUMSPs)
Input: A MTS-Tree T, a header table H, a minimum utility threshold ε,
and a minimum support threshold δ
Output: A WUMSP-Tree T'
1.  Let T' be an WUMSP-Tree
2.  foreach WULI α in the bottom entry of H do
3.    trace the link of WULI α in H to get 1-WULP
4.    add 1-WULP α and sid to T'
5.    create a conditional MTS-Tree CT_α and a header table H_α
6.    call WUMSP-Mine(CT_α, H_α, α)

Procedure WUMSP-Mine(CT_α, H_α, α)
1.  foreach WULI β in H_α do
2.    if sup(β)< δ or SWU(β)< ε then
3.      delete β from CT_α and H_α
4.      if there exists an empty node X in CT_α
5.        delete X
6.        append the X's children nodes to X's parent node
7.  foreach WULI β of HT_α do
8.    add WULP βα and its sids to T'
9.    trace the paths of βα in T
10.   calculate the corresponding supports and SWUs
11.   add the paths to the path table of βα in the node of β in T'
    /* line 12-14: Path pre-checking technique*/
12.   if there exists a path in βα to form a WUMSP Y, such that
    sup(Y)≧ δ and SWU(Y)≧ ε then
13.     create a conditional MTS-Tree CT_βα and a header table H_βα
14.     call WUMSP-Mine(CT_βα, H_βα, βα)
```

Fig. 4. The procedure of generating WULPs



(a) Conditional MTS-tree of <H; t_7> (b) WUMSP-Tree

Fig. 5. Conditional MTS-Tree of <H; t7> and the corresponding WUMSP-Tree

By tracing from a node to root in a WUMSP-Tree, a WULP can be derived. Furthermore, the corresponding WUMSPs of the WULP can be obtained after combining the paths in the node. Take the WUMSP-Tree in Figure 5 for example, we can get a WULP <G;t_6><H;t_7> by tracing from the node to root. Moreover, we can get a WUMSP <{<G;t_6><H;t_7>}; GLH> by combining the path GLH in the node <G;t_6> with the WULP. After tracing all nodes in WUMSP-Tree, all WUMSPs can be obtained. By storing the WUMSPs in the WUMSP-Tree, the patterns can be compressed in the tree and the memory storage can be saved.

During the processes of generating WULPs, if the length of the WULPs is larger than 1, besides the processes of tracing path, a *path pre-checking technique* will be performed to prune the moving patterns which can not fit the user-specified thresholds.

Definition 10. (Path pre-checking technique): If there exists no path in a WULP X to form a WUMP Y such that $sup(Y) \geq \delta$ and $SWU(Y) \geq \varepsilon$, X is pruned.

Path pre-checking technique is used for trimming the search space. By using this technique, the number of conditional MTS-Tree can be reduced effectively and the mining performance can be more improved.

Now we introduce the processes of the step 3, i.e., generating WUMSPs from MTS-Tree, by continuing the example in the previous section. First, the last entry <H;t_7> in the header table of the MTS-Tree shown in Figure 3 is checked and a WULP <H;t_7> is generated. Then <H;t_7> and its SIDs are inserted as the first child node of the root of the WUMSP-Tree. Since <H;t_7> is a 1-WULP, its path is not traced. Then the conditional MTS-Tree of <H;t_7> shown in Figure 5 is constructed by tracing all ancestor nodes of the nodes labeled <H;t_7> in MTS-Tree. The nodes labeled <H;t_7> can be acquired by tracing the links from the entry of <H;t_7> in header table. Note that the conditional MTS-Trees do not need any path table.

Subsequently, in the header table of the conditional MTS-Tree of <H;t_7>, the last entry <G;t_6> is checked and a WULP <G;t_6><H;t_7> is generated and inserted into the WUMSP-Tree. Since there is already a node <H;t_7> in the WUMSP-Tree, we just insert <G;t_6> as a child node of <H;t_7>. At the same time, the path of the WULP <G;t_6><H;t_7> is traced in the original MTS-Tree. By the links from the entry <H;t_7> in header table, we can get the node <H[t_7]> with the SIDs S_2 and S_4. The node <H[t_7]> is traced up until the node <G;t_6> is reached to obtain the paths between the nodes. Then a path GLH is found. After tracing the path, a WUMSP <{<G;t_6><H;t_7>}; GLH> whose support equals to 2 and SWU equals to 191 is found. By the path pre-checking technique, since its support and SWU are both no less than the two thresholds, it is kept, and the path is added into the node <G;t_6> of the WUMSP-Tree. The WUMSP-Tree now is shown in Figure 5. Generating the patterns from the MTS-Tree by the above processes recursively, all WUMSPs can be generated.

4.1.3 Finding high utility mobile sequential patterns

After generating all WUMSPs, an additional database scan will be performed to find UMSPs from the set of WUMSPs. The WUMSPs whose utilities are larger than or equal to the minimum utility threshold will be regarded as UMSPs. Moreover, since the WUMSPs in WUMSP-Tree include SIDs, instead of checking all mobile transaction sequences, they will just check the specified sequences. By applying this process, the mining performance will become better.

4.2 An Improved Tree-based Method: $UMSP_{BFG}$

In $UMSP_{DFG}$, since the number of combinations of 2-WULPs is quite large, many conditional MTS-Trees will be generated. Dealing with these conditional MTS-Trees is a hard work in the mining processes. Moreover, tracing the paths of WULPs in the processes of generating WUMSPs also consumes much time. If we can decrease the

number of WULPs requiring verification, especially the large number of 2-WULPs, the performance can be more improved. Therefore, how to speed up the processes about 2-WUMSPs is a crucial problem.

To conquer this problem, we propose an improved tree-based algorithm $UMSP_{BFG}$ (*high Utility Mobile Sequential Pattern mining with a tree-based Breadth First Generation strategy*). The difference between the two algorithms is that $UMSP_{BFG}$ use a breadth first generation strategy for generating 2-WUMSPs. Within the strategy, a *possible succeeding node checking technique* is applied. By this technique, the size of the conditional MTS-Trees will be smaller, and the 2-moving patterns which cannot be 2-WUMSPs will be pruned in advance.

Instead of generating a 2-WULP by combining the last entry with the 1-WULP of a conditional MTS-Tree, in the breadth first generation strategy, 2-WULPs are generated by combining all 1-WULPs in the header table with the 1-WULP of the conditional MTS-Tree. After generating the 2-WULPs, their paths, supports and SWUs are checked in advance. The valid paths will be stored in the corresponding nodes of WUMSP-Tree. While generating 2-WULPs, $UMSP_{BFG}$ applies a *possible succeeding node checking technique* for pruning useless 2-WULPs, which is addressed as follows.

Definition 11. (Possible succeeding node checking technique) While generating 2-WULPs of a 1-WULP X in X's conditional MTS-Tree, all 1-WULPs in the header table are inserted as children nodes of X in the WUMSP-Tree in advance. If there exists no path in a WULP Y to form a WUMSP Z such that $sup(Z) \geq \delta$ and $SWU(Z) \geq \varepsilon$, Y is pruned. Furthermore, only the nodes kept in the WUMSP-Tree are able to be succeeding nodes of the WUMSP-Tree in the later mining processes.

In the following paragraphs, we use the same example as the previous section. The MTS-Tree shown in Figure 3 and the conditional MTS-Tree of <H;t_7> shown in Figure 5 are constructed by the same processes in previous section. <H;t_7> is inserted into the WUMSP-Tree as the first node. Different from $UMSP_{DFG}$, in the processes of generating 2-WULPs of $UMSP_{BFG}$, all 1-WULPs in the header table of the conditional MTS-Tree of <H;t_7> are inserted as children nodes of the node <H;t_7> in the WUMSP-Tree, that is, all 2-WULPs of the conditional MTS-Tree of <H;t_7> are generated in advance. The paths of the 2-WULPs are then generated by tracing the original MTS-Tree. Combining the 2-WULPs and the paths, 2-moving patterns are generated. Also, their supports and SWUs are obtained. The results are shown in Figure 6.

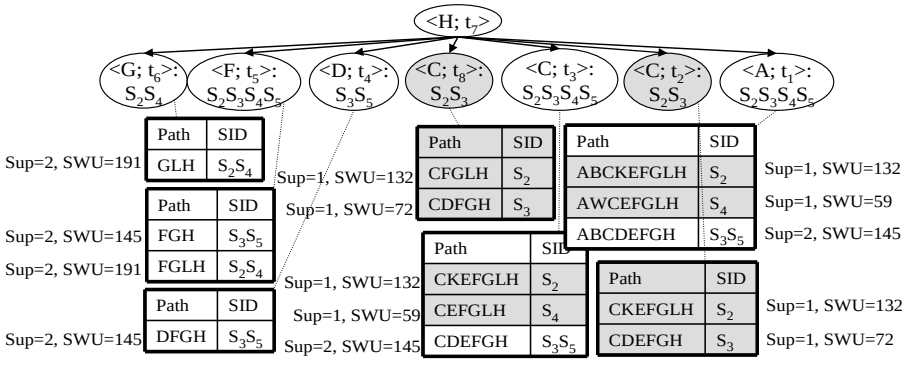


Fig. 6. An example of WUMSP-Tree generated by $UMSP_{BFG}$.

By Figure 6, since the supports or SWUs of the 2-moving patterns <{<C;t_8><H;t_7>}; CFGLH>, <{<C;t_8><H;t_7>};CDFGH>, <{<C;t_3><H;t_7>};CKEFGLH>, <{<C;t_3><H;t_7>};CEFGLH>, <{<C;t_2><H;t_7>};CKEFGLH>, <{<C;t_2><H;t_7>};CDEFGH>, <{<A;t_1><H;t_7>};ABCKEFGLH> and <{<A;t_1><H;t_7>};AWCEFGLH> are less than the thresholds, their relevant paths are pruned from the path tables of the WUMSP-Tree. Moreover, since there is no valid path in the nodes <C;t_8> and <C;t_2>, the two nodes are also pruned. In Figure 6, the pruned nodes and paths in the WUMSP-Tree are labeled with grey. By the WUMSP-Tree, we can know the possible succeeding nodes of <G;t_6> are <F;t_5>, <D;t_4>, <C;t_3> and <A;t_1>.

After ascertaining which 2-moving patterns need to be pruned, the relevant nodes and entries in the conditional MTS-Tree of <H;t_7> are also pruned. After this step, the mining processes proceed without the pruned nodes in both the WUMSP-Tree and the conditional MTS-Tree of <H;t_7>. The remaining conditional MTS-Tree is much smaller than the original one. Moreover, since the useless entries are pruned in the header table, they will never be checked in the following processes. Therefore, the search space can be further reduced and the mining performance is further improved.

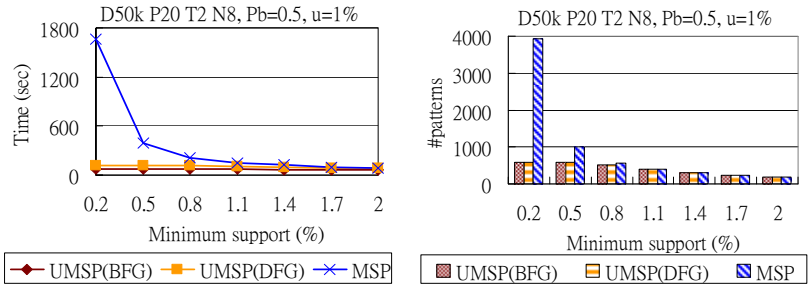
5 Experimental Results

In this section, we evaluate the performance of the proposed algorithms. The experiments were performed on a 2.4 GHz Processor with 1.6 GB memory, and the operating system is Microsoft Windows Server 2003. The algorithms are implemented in Java. The default settings of the parameters are listed in Table 4. The settings of parameters related to mobile commerce environment and utility mining are similar to [14] and [8], respectively.

For comparing the performance of the proposed algorithms, we extend algorithm TJ_{PF} in [14] to form a basic algorithm for mining UMSPs which is called MSP in this paper. The processes of MSP are as follows: First, the mobile sequential patterns whose supports are no less than the minimum support threshold are generated by TJ_{PF}. Then an additional check of the actual utilities of the mobile sequential patterns is performed for finding UMSPs. In the following experiments, the performance of MSP is compared with that of the two proposed algorithms. Due to the page limit, in the experiment results, we show the number of patterns after phase I instead of the execution time of phase II since the time cost is mainly decided by the number of these patterns. The fewer patterns should be checked, the less time will be spent.

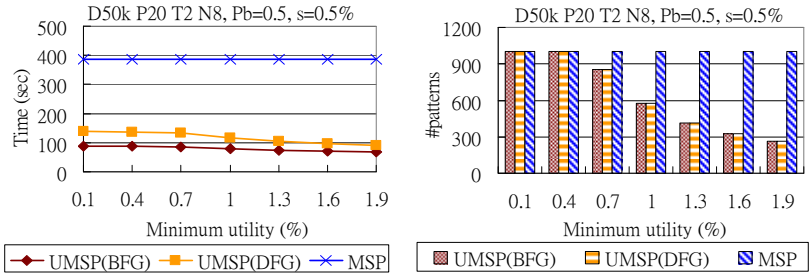
Table 4. Parameter settings

Parameter Descriptions	Default
D: Number of mobile transaction sequences	50k
P: Average length of mobile transaction sequences	20
T: Average number of items per transaction	2
N: Size of mesh network	8
$\mathbf{n_I}$: The range of the number of items sold in each location	200
$\mathbf{P_b}$: The probability that user makes the transaction in the location	0.5
w: Unit profit of each item	1~1000
q: Number of purchased items in transactions	1~5



(a) Execution time (b) Number of patterns after Phase I

Fig. 7. The performance under varied minimum support thresholds.

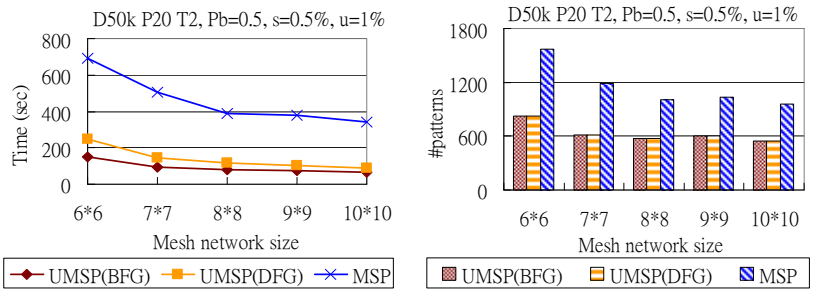


(a) Execution time (b) Number of patterns after Phase I

Fig. 8. The performance under varied minimum utility thresholds.

The first part of the experiments is the performance under various minimum support thresholds. In the experiments, the minimum utility threshold is set as 1%. The results for the execution time and the number of patterns after phase I under varied minimum support thresholds are shown in Figure 7. For the two proposed algorithms, the patterns after phase I are WUMSPs, on the other hand, for MSP, the patterns are mobile sequential patterns. In Figure 7 (a), it can be seen that MSP requires much more execution time than the other algorithms. The reason is that since MSP does not consider utility in phase I, the number of generated patterns is much larger than that of other algorithms as shown in Figure 7 (b). MSP spends much more time on processing additional patterns, so its performance is the worst. Besides, although the number of WUMSPs generated by the proposed algorithms is the same, their execution time is different. Overall, the tree-based algorithms are better than the level-wise version especially when the minimum support threshold is low.

The second part of the experiments is the performance under various minimum utility thresholds. In the experiments, the minimum support threshold is set as 0.5%. The results are shown in Figure 8. Overall, the tree-based algorithms are better than the level-wise one. Besides, since MSP does not consider utility in phase I, its execution time and number of generated patterns remain the same. On the contrary, both of the two results of the proposed two algorithms decrease with the minimum utility threshold increasing. In Figure 8 (b), when the minimum utility threshold is below 0.4%, almost no candidate can be pruned. Thus, the performance of the two algorithms is almost the same.



(a) Execution time (b) Number of patterns after phase I

Fig. 9. The performance under varied mesh network size.

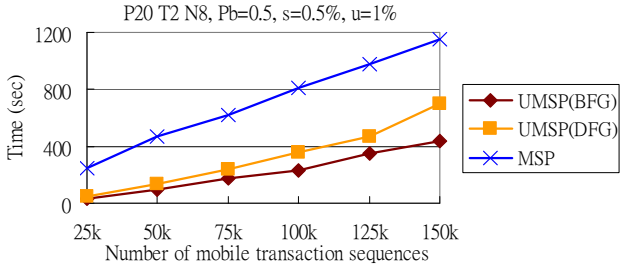


Fig. 10. The execution time under varied number of mobile transaction sequences.

The third part of the experiments is the performance under varied mesh network size. The results are shown in Figure 9. By Figure 9, it can be seen that the execution time of all the three algorithms decreases with the size of mesh network. The reason is that when the size of mesh network is larger, the database will be sparser, therefore, the patterns generated in phase I will become fewer and the time cost of mining processes will be reduced.

The final part of the experiments is the performance under varied number of mobile transaction sequences. The experimental results are shown in Figure 10. In this figure, we can see that when the number of mobile transaction sequences is larger, the execution time of the algorithms increases linearly.

By the above experiments, the proposed algorithms are shown to outperform the state-of-the-art mobile sequential pattern algorithm MSP. Among the algorithms, the performance of $UMSP_{BFG}$ is the best since the MTS-tree is an efficient tree structure and the breadth first strategy effectively enhances the mining performance.

6 Conclusions

In this research, we proposed a novel data mining issue about mining high utility mobile sequential patterns in mobile commerce environments. This paper is the first research work about the combination of mobility pattern mining and utility mining. Two algorithms developed by different strategies, i.e., depth first generation and

breadth first generation, are proposed for efficiently mining high utility mobile sequential patterns. The experimental results show that the proposed algorithms outperform the state-of-the-art mobile sequential pattern algorithm. For future work, additional experiments under more conditions of mobile commerce environments will be conducted for further evaluating the algorithms. Moreover, new algorithms which improve the mining performance will be designed.

Acknowledgments. This research was supported by National Science Council, Taiwan, R.O.C. under grant no. NSC99-2631-H-006-002 and NSC99-2218-E-006-001.

References

1. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. of the 20th Int'l. Conf. on Very Large Data Bases, pp. 487-499, 1994.
2. R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. of 11th Int'l. Conf. on Data Mining, pp. 3-14, 1995.
3. C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," IEEE Trans. on Knowledge and Data Engineering, vol. 21, issue 12, pp. 1708-1721, 2009.
4. R. Chan, Q. Yang, and Y. Shen, "Mining High Utility Itemsets," Proc. of Third IEEE Int'l Conf. on Data Mining, pp. 19-26, Nov., 2003.
5. J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. of the ACM-SIGMOD International Conference on Management of Data, pp. 1-12, 2000.
6. 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. Journal of Computer Science Security, vol. 7, no. 2, pp. 285-291, 2007.
7. Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," Data & Knowledge Engineering, vol.64, issue 1, pp.198-217, 2008.
8. Y. Liu, W.-K. Liao and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," Proc. of Utility-Based Data Mining, 2005.
9. E. H.-C. Lu, and V. S. Tseng, "Mining Cluster-based Mobile Sequential Patterns in Location-based Service Environments," Proc. of IEEE Int'l. Conf. on Mobile Data Management, 2009.
10. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M. C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach," IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 10, Oct. 2004.
11. V. S. Tseng and W. C. Lin, "Mining Sequential Mobile Access Patterns Efficiently in Mobile Web Systems," Proc. of the 19th Int'l. Conf. on Advanced Information Networking and Applications, pp. 867-871, Taipei, Taiwan, 2005.
12. V. S. Tseng, C. W. Wu, B.-E. Shie, and P. S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," Proc. of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2010), Washington, DC, USA, Jul. 2010.
13. S.-J. Yen and Y.-S. Lee, "Mining High Utility Quantitative Association Rules," Proc. of 9th Int'l Conf. on Data Warehousing and Knowledge Discovery, Lecture Notes in Computer Science 4654, pp. 283-292, Sep. 2007.
14. C.-H. Yun and M.-S. Chen, "Mining Mobile Sequential Patterns in a Mobile Commerce Environment," IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, vol. 37, no. 2, 2007.