Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 1971805, 11 pages
https://doi.org/10.1155/2020/1971805

Research Article

Improved Strategy for High-Utility Pattern Mining Algorithm

Le Wang,[1] Shui Wang,[1] Haiyan Li,[2] and Chunliang Zhou[1]

[1]*College of Digital Technology and Engineering, Ningbo University of Finance & Economics, Ningbo, Zhejiang 315175, China*
[2]*School of Information Science and Engineering, Yunnan University, Kunming, Yunnan 650050, China*

Correspondence should be addressed to Shui Wang; seawan@163.com

Received 20 July 2020; Revised 2 November 2020; Accepted 16 November 2020; Published 26 November 2020

Academic Editor: Sergio Ortobelli

High-utility pattern mining is a research hotspot in the field of pattern mining, and one of its main research topics is how to improve the efficiency of the mining algorithm. Based on the study on the state-of-the-art high-utility pattern mining algorithms, this paper proposes an improved strategy that removes noncandidate items from the global header table and local header table as early as possible, thus reducing search space and improving efficiency of the algorithm. The proposed strategy is applied to the algorithm EFIM (EFficient high-utility Itemset Mining). Experimental verification was carried out on nine typical datasets (including two large datasets); results show that our strategy can effectively improve temporal efficiency for mining high-utility patterns.

1. Introduction

The main challenge of data mining is to find meaningful information from massive amounts of data. The technique of finding interesting, unexpected, and useful data patterns from large databases is called pattern mining. Many studies have focused on traditional frequent pattern mining and just concern the occurrence of itemsets/patterns in the database, without considering the internal utility values (i.e., quantity) and external utility values (e.g., importance, profit, and price) of each item in the itemset [1]. To address this issue, the utility information of each item or itemset is introduced into frequent pattern mining, hence the emergence of high-utility patterns/itemsets (HUPs/HUIs) mining. HUP mining has been used in many fields, unfolding its commercial value in many application fields, such as website clickstream analysis [2, 3], mobile commerce environment [4], cross-marketing commercial value of retail stores [5, 6], and gene regulation and biomedical applications [7]. Utility patterns are also applied on sequential data, such as algorithm HUSP-ULL [8], and uncertain data, such as algorithm MUHUI [9].

Yao et al. [10] proposed the definition and mathematical model of high-utility pattern (HUP): the utility value $U(X)$ of an itemset X on a dataset D is defined as the sum of the utility value $U(X, t)$ of X on all transactions t (see Definition 3). The task of the high-utility pattern mining is to find all patterns whose utility value is not less than the minimum utility value (threshold value). The pruning strategy of traditional frequent pattern mining algorithms no longer works in HUP mining, because a superset of a non-HUP might be an HUP; this makes the search space of the mining algorithm even larger. Improving the spatial-temporal efficiency of the mining algorithm has been a challenge [11–13].

Existing HUPs mining algorithms may be categorized into a two-phase approach and one-phase approach. Two-phase mining algorithms need two phases to find all HUPs: in the first phase, the candidate itemsets are generated by estimating the utility value of each candidate itemset, in the second phase, the true utility value of each candidate itemset is calculated by scanning the dataset. This two-phase approach is adopted by the algorithms of Two-Phase [11], IHUP [2], UP-Growth [14], and MU-Growth [15]. These algorithms often generate a large number of candidate itemsets in the first phase, not only requiring much of storage memory but also drastically increasing the computation cost in the second phase.

In order to avoid the problems caused by candidate itemsets, newly proposed HUP mining algorithms tend to use a no-candidate approach, such as HUI-Miner [16] and d2HUP [17]; they try to find HUPs directly without generating candidate itemsets. Comparing with the two-phase HUP mining algorithms, the speeds of algorithms d2HUP and HUI-Miner are greatly improved. Based on HUI-Miner, two improved algorithms HUP-Miner [12] and FHM [18] are developed. Also, there is a new fast mining algorithm EFIM [19] proposed by Zida et al., in which two new upper bounds of utility value are used to reduce the search space and consequently boost the performance greatly. The algorithm HMiner [20] proposed two pruning techniques, LAprune and C-prune, to reduce the search space for mining HUPs. The algorithm ULB-Miner [21] extended the algorithms FHM [18] and HUI-Miner [16] by utilizing a utility list buffer structure, which improved the performance of the FHM algorithm in terms of the memory and runtime usage.

Although the spatial-temporal efficiency of HUP mining algorithm has been greatly improved, the time cost is still relatively high. The improvement of algorithm performance, especially the improvement of its temporal efficiency, is still a challenge in this field. In this paper, an improved strategy is proposed to boost the temporal efficiency of HUP mining algorithm and is applied to the algorithm EFIM.

The rest of this paper is organized as follows. Section 2 introduces the problem description and relevant definitions. Section 3 introduces the improvement strategy and the improved algorithm EFIM-IMP. Section 4 reports the experimental results. Section 5 draws the conclusions.

2. Problem Description and Definitions

A *utility-valued transaction dataset* $D = \{T_1, T_2, T_3, \ldots, T_n\}$ contains n transactions and m unique items $I = \{i_1, i_2, \ldots, i_m\}$. Each transaction T_d $(d = 1, 2, 3, \ldots, n)$ is called a transaction itemset and includes a subset of all unique items in I, for example, $T_1 = \{(A,4)\ (C,3)\ (F,1)\}$. Each item i_j in each transaction T_d is attached with a quantity which is called *internal utility* (denoted as $q(i_j,T_d)$); for example, the first transaction in Table 1 includes 3 items “A,” “C,” and “F”; their quantities are 4, 3, and 1, respectively, denoted as $q(A,T_1) = 4$, $q(C,T_1) = 3$, and $q(F,T_1) = 1$. Each item i_j has a unit profit $p(i_j)$, which is called *external utility*, for example, $p(A) = 4$ in Table 2. $|D|$ indicates the number of transactions in the dataset D, and $|T_d|$ indicates the number of items in the transaction T_d.

Definition 1. The utility value of the item i_j in a transaction T_d is denoted as $U(i_j, T_d)$ and is defined as

$$U(i_j, T_d) = p(i_j) * q(i_j, T_d). \tag{1}$$

For example, in Tables 1 and 2, $U(A,T_1) = 4 * 4 = 16$, $U(C,T_1) = 10 * 3 = 30$, and $U(F,T_1) = 1 * 1 = 1$.

Definition 2. The utility value of the itemset X in a transaction T_d is denoted as $U(X, T_d)$ and is defined as

$$U(X, T_d) = \begin{cases} \sum_{i_j \in X} U(i_j, T_d), & \text{if } X \subseteq T_d, \\ 0, & \text{else.} \end{cases} \tag{2}$$

Table 1: A transaction dataset D.

TID	Transaction	*TU*
T_1	(A, 4) (C, 3) (F, 1)	47
T_2	(C, 1) (D, 4) (E, 10)	58
T_3	(A, 4) (B, 4) (D, 2) (E, 6)	54
T_4	(A, 6) (B, 2) (D, 2) (E, 1)	46
T_5	(A, 3) (B, 3) (D, 1) (G, 1)	30
T_6	(A, 3) (B, 7) (C, 1) (E, 3)	49

Table 2: A profit table.

Item	Profit
A	4
B	3
C	10
D	7
E	2
F	1
G	2

For example, in Tables 1 and 2, $U(\{AC\}, T_1) = 46$, $U(\{AF\}, T_1) = 17$.

Definition 3. The utility value of the itemset X in a dataset D is denoted as $U(X)$ and is defined as

$$U(X) = \sum_{T_d \in D \wedge X \subseteq T_d} U(X, T_d). \tag{3}$$

For example, in Tables 1 and 2, $U(\{AC\}) = U(\{AC\}, T_1) + U(\{AC\}, T_6) = 46 + 22 = 68$.

Definition 4. The utility value of the transaction T_d in a dataset D is denoted as $TU(T_d)$ and is defined as

$$\mathrm{TU}(T_d) = \sum_{i_j \in T_d} U(i_j, T_d). \tag{4}$$

For example, in Tables 1 and 2, $TU(T_1) = U(A, T_1) + U(C, T_1) + U(F, T_1) = 16 + 30 + 1 = 47$.

Definition 5. The utility value of the dataset D is denoted as TU and is defined as

$$\mathrm{TU} = \sum_{T_d \in D} \mathrm{TU}(T_d). \tag{5}$$

For example, in Tables 1 and 2, $TU = 47 + 58 + 54 + 46 + 30 + 49 = 284$.

Definition 6. The *transaction-weighted utility value* of the itemset X is denoted as $TWU(X)$ (also called TWU value) and is defined as

$$\mathrm{TWU}(X) = \sum_{T_d \in D \wedge T_d \supseteq X} \mathrm{TU}(T_d). \tag{6}$$

Definition 7. The *minimum utility threshold* δ is a user-specified percentile of the total transaction utility value of the given dataset D; so the *minimum utility value*, *MinU*, in D is defined as

$$\operatorname{Min} U = \mathrm{TU} * \delta. \tag{7}$$

Definition 8. An itemset X is called a *high-utility pattern/itemset* (HUP/HUI) if its utility is not less than the *minimum utility value.*

Definition 9. An itemset/item X is called a *candidate itemset/item* for high-utility itemset/item if $twu(X) \geq \operatorname{Min} U$, and it is also called a *promising itemset/item*; otherwise it is an *unpromising itemset/item.*

Theorem 1. *Transaction-weighted downward closure property [4]: any nonvoid subset of a promising itemset is a promising itemset, and any superset of an unpromising itemset is an unpromising itemset.*

Definition 10. Assume that the transaction T_d in dataset D is ordered (e.g., in lexicographic order), for item i_j, the subsequent items i_k $(k \geq j)$ are the *remaining itemset* of item i_j, denoted as $RI(i_j, T_d)$.

Definition 11. The *utility value of the remaining itemset* (i_j, T_d) in transaction T_d is denoted as $RU(i_j, T_d)$ and is defined as

$$\mathrm{RU}(i_j, T_d) = \sum_{i \in \mathrm{RI}(i_j, T_d)} U(i, T_d). \tag{8}$$

For example, items of each transaction in Table 1 are ordered lexicographically, so $RU(\mathrm{C}, T_1) = U(\mathrm{C}, T_1) + U(\mathrm{F}, T_1) = 31$, $RU(\mathrm{C}, T_2) = U(\mathrm{C}, T_2) + U(\mathrm{D}, T_2) + U(\mathrm{E}, T_2) = 10 + 28 + 20 = 58$, and $RU(\mathrm{C}, T_6) = U(\mathrm{C}, T_6) + U(\mathrm{E}, T_6) = 10 + 6 = 16$.

Definition 12. The *utility value of the remaining itemset* of item i_j in dataset D is denoted as $RU(i_j)$ and is defined as

$$\mathrm{RU}(i_j) = \sum_{T_d \in D \wedge i_j \in T_d} \mathrm{RU}(i_j, T_d). \tag{9}$$

For example, for Tables 1 and 2, $RU(\mathrm{C}) = 31 + 58 + 16 = 105$.

Theorem 2. *For an itemset/item $\{i_j\}$, if $RU(i_j) < MinU$, then it is not an HUP, and its any superset Y $(Y \in \{RI(i_j, T_d)|, \forall \{i_j\} \subset T_d\})$ is not an HUP either.*

Proof. According to Definitions 3 and 12, $RU(i_j) > U(i_j)$ and $RU(i_j) > U(Y)$; so itemset $\{i_j\}$ or Y is not an HUP if $RU(i_j) < MinU$. □

3. Algorithm EFIM and the Improved Version EFIM_IMP

3.1. Algorithm EFIM. The algorithm EFIM uses a pattern-growth approach to find HUPs; the main process is to find candidate items by scanning the dataset, and then iteratively generates new candidate items by scanning the local dataset of each candidate. We can see from the above that the fewer of the candidates, the less search space is needed in the iterative process, and the more efficient the algorithm will be.

So EFIM proposes two upper bounds of utility value of HUP, and apply them respectively to the global and local dataset, to tighten the criteria of candidates and to reduce the number of candidate items generated, resulting more efficient mining algorithm. The algorithm EFIM is shown as Algorithms 1 and 2.

Lines 1–8 in Algorithm 1 process the global dataset for all candidates with the *remaining utility value* not less than the *minimum utility threshold*. Lines 1–3 calculate *twu* values of each item (lu (α,i)), and candidates (whose *twu* values are not less than the *minimum threshold*) are stored to list *Secondary* (α). Line 4 sorts the items in *Secondary* (α) by ascending order of *twu* values. Line 5 deletes noncandidates from each transaction, sorts items in each transaction according to the order of list *Secondary* (α), and removes empty transactions. Line 6 sorts all transactions and merges the transactions with the same items. Line 7 calculates the *remaining utility value* of each item in transactions. By Line 8, items with *remaining utility value* note less than the *minimum utility* are stored to list *Primary* (α). Line 9 iteratively processes each candidate in *Primary* (α) and determines if it (and its extended itemset) is an HUP.

Detailed procedures of Line 9 (in Algorithm 1, for iteratively processing the local dataset) are shown in Algorithm 2, as a subroutine named *Search*. Lines 1–9 deal with each item in *Primary* (α). Line 3 scans dataset α–D (that contains itemset α), calculates utility value of itemset β, and gets dataset β-D (that contains itemset β). Line 4 outputs β as an HUP if its utility value is not less than the *minimum threshold*. Lines 5–7 scan β-D and get *Primary*(β) and *Secondary*(β) using the same mechanism as Figure 1 for *Primary* (α) and *Secondary* (α). Line 8 iteratively calls *Search* for iteration on *Primary* (β).

By utilizing two upper bounds on utility value, EFIM can effectively reduce the number of candidates (candidate-items) and boost the performance of the mining algorithm. But the number of candidates still can be reduced in algorithms like EFIM, so we propose two additional strategies to further reduce the number of candidates.

3.2. Improvement of Algorithm EFIM. There is a fact that EFIM does not take into consideration: when deleting noncandidate items from the dataset, the *TWU* values of candidate items might be affected and reduced, resulting in some of the candidate items to be changed to noncandidates. This is an iterative process until the dataset goes stable for a

```
Input: $D$: a transaction database; $MinU$: a user-specified threshold.
Output: the set of high-utility itemsets
(1) $\alpha = \varnothing$
(2) Calculate $lu(\alpha,i)$ for all items $i \in I$ by scanning $D$, using a utility-binary;
(3) $Secondary(\alpha) = \{i | i \in I \wedge lu(\alpha,i) \geq MinU\}$;
(4) Let $\succ$ be the total order of TWU ascending values on $Secondary(\alpha)$;
(5) Scan $D$ to remove each item $i \notin Secondary(\alpha)$ from the transactions, sort items in each transaction according to $\succ$, and delete
    empty transactions;
(6) Sort transactions in $D$ according to $\succ_T$;
(7) Calculate the subtree utility $su(\alpha,i)$ of each item $i \in Secondary(\alpha)$ by scanning $D$, using a utility-bin array;
(8) $Primary(\alpha) = \{i | i \in Secondary(\alpha) \wedge su(\alpha,i) \geq MinU\}$;
(9) Search ($\alpha$, $D$, $Primary(\alpha)$, $Secondary(\alpha)$, $MinU$);
```

Algorithm 1: The EFIM algorithm.

```
Input: $\alpha$: an itemset; $\alpha$-$D$: the $\alpha$ projected database; $Primary(\alpha)$: the primary items of $\alpha$; $Secondary(\alpha)$: the secondary items of $\alpha$;
$MinU$: a user-specified threshold.
Output: the set of high-utility itemsets that are extensions of $\alpha$
(1) for each item $i \in Primary(\alpha)$ do
(2)     $\beta = \alpha \cup \{i\}$;
(3)     Scan $\alpha-D$ to calculate $u(\beta)$ and create $\beta-D$//uses transaction merging;
(4)     if $u(\beta) \geq MinU$ then output $\beta$;
(5)     Calculate $su(\beta,z)$ and $lu(\beta,z)$ for all item $z \in Secondary(\alpha)$ by scanning $\beta-D$ once, using two utility-bin arrays;
(6)     $Primary(\beta) = \{z \in Secondary(\alpha) | su(\beta,z) \geq MinU\}$;
(7)     $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta,z) \geq MinU\}$;
(8)     Search ($\beta$, $\beta-D$, $Primary(\beta)$, $Secondary(\beta)$, $MinU$);
(9) end for
```

Algorithm 2: The search procedure.

specified minimum *TWU* threshold. We harness this fact for an improved strategy to reduce the number of candidate items effectively, applying this strategy to the candidate generating process of the EFIM algorithm from both the original (global dataset) and local datasets.

3.2.1. Improved Strategy on Global Candidate Itemset. Algorithm EFIM calculates *TWU* value of each item (Line 2 in Algorithm 1) and filters out those items with *TWU* values less than the minimum threshold from the original transaction dataset, leaving the rest of the items as the *candidate items*. If we recalculate the *TWU* value of each item again after the deletion, the new *TWU* values might be less than their original counterparts; that is, some of the candidate items might no longer reach the criteria of minimum *TWU* threshold. We can delete the newly established noncandidates to reduce the computational cost hereafter and increase the time-space efficiency of the algorithm.

So, we propose our first improving strategy.

Strategy 1. Repeatedly calculate *TWU* values and delete noncandidate items from the global header table, till the recalculation does not generate new noncandidate items.

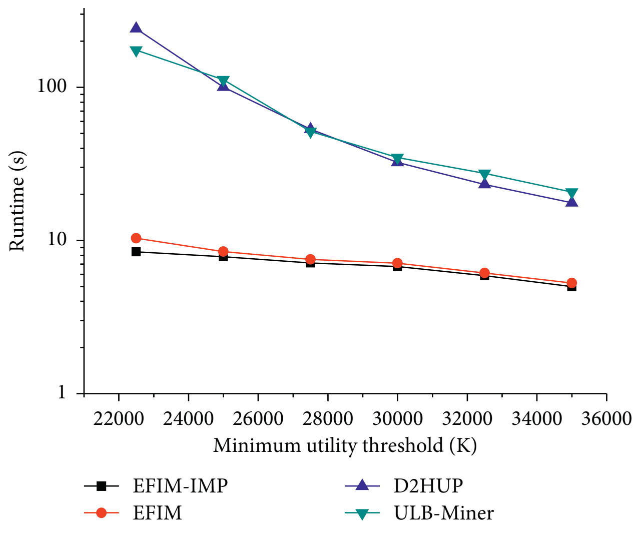
The purpose of Strategy 1 is to reduce the number of candidates in the global header table and hence reduce the search space of the algorithm hereafter. Applying this strategy to EFIM's global dataset processing, we get the improved algorithm EFIM-IMP in Algorithm 3.

Lines 1–3 of EFIM-IMP (Algorithm 3) function the same way as EFIM: scan the dataset once, get each item's *TWU* value, and put those items with *TWU* values not less than the minimum threshold to *Secondary* (α).

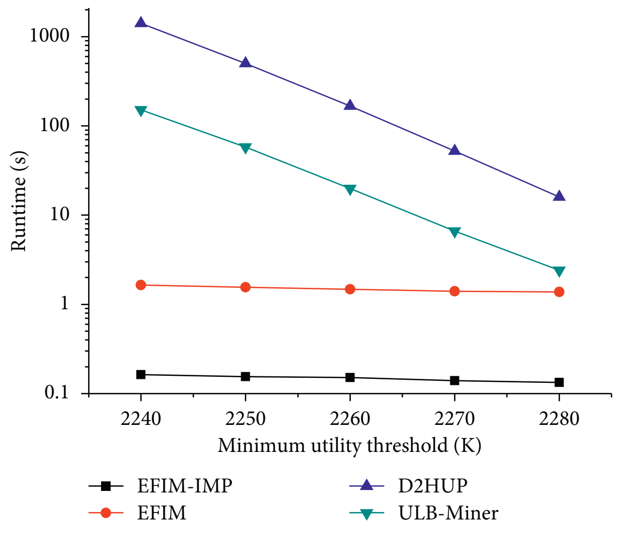
Lines 4–12 (Algorithm 3) are our improved strategy: Line 4 counts the number of unique items in the original dataset to *count0*; Line 5 counts the number of candidates to *count1*; Lines 6–12 are repeated deletions of noncandidates, as long as the number of candidates changes after a deletion; Line 7 deletes all noncandidate items from the dataset; Line 8 recalculates the *TWU* value of each item; Line 9 recounts candidates to *Secondary* (α); Line 10 saves the count of candidates of the last iteration; and Line 11 gets the count of the remaining candidates.

This iterative deleting strategy reduces the number of items in *Primary* (α) and *Secondary* (α) and hence reduces the search space of algorithm EFIM.

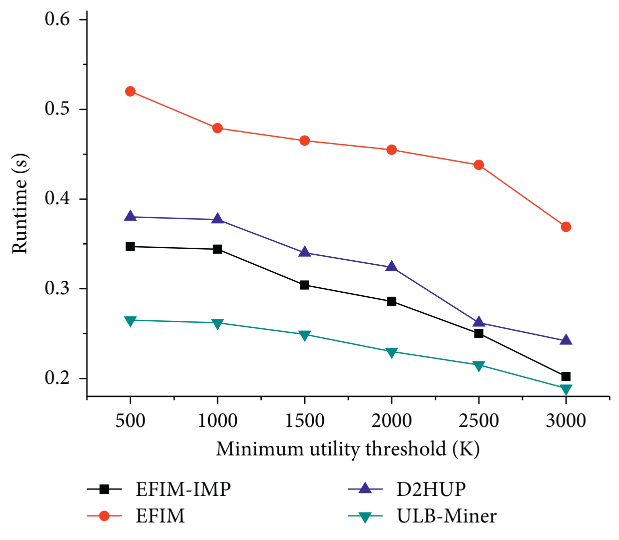
3.2.2. Improved Strategy on Local Candidate Itemset. The algorithm in Algorithm 1 mainly deals with the original dataset, resulting in two lists *Primary* (α) and *Secondary* (α); these lists are called the global candidate lists. Algorithm 2 deals with the subset of a certain item/itemset β (called a



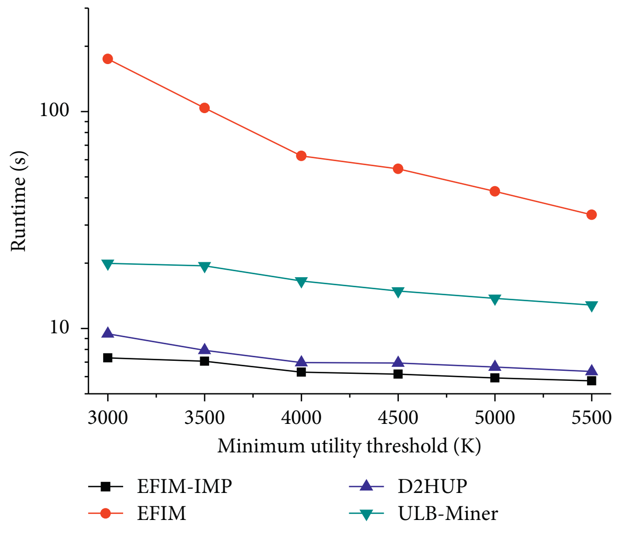
(a)



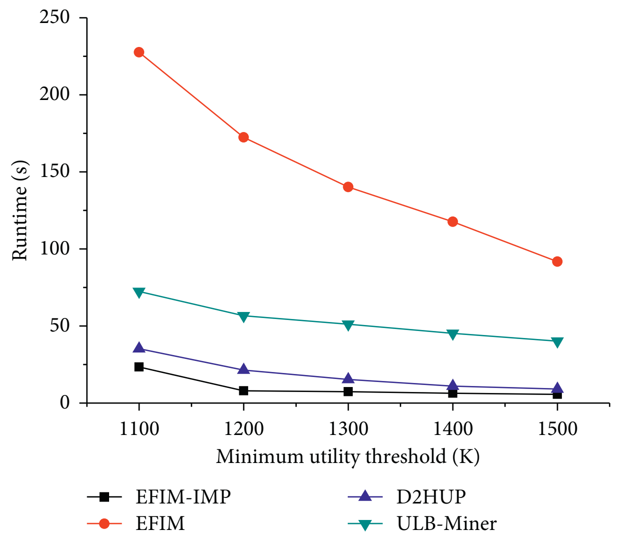
(b)



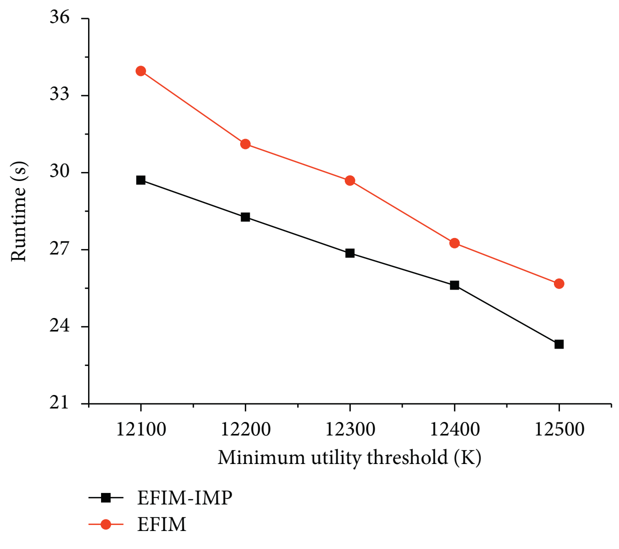
(c)



(d)

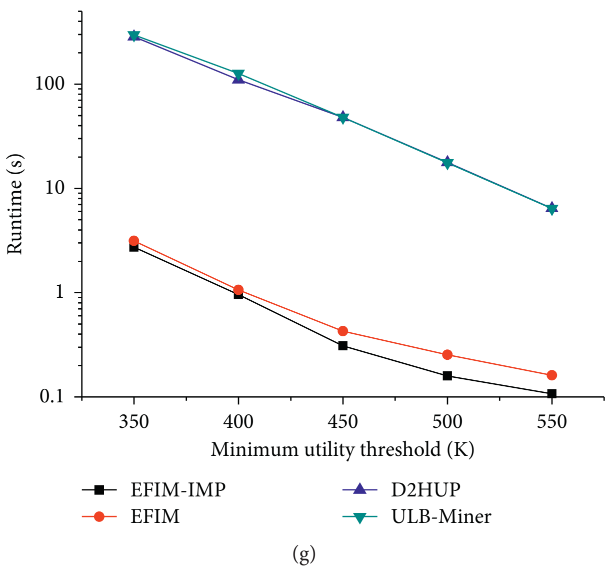


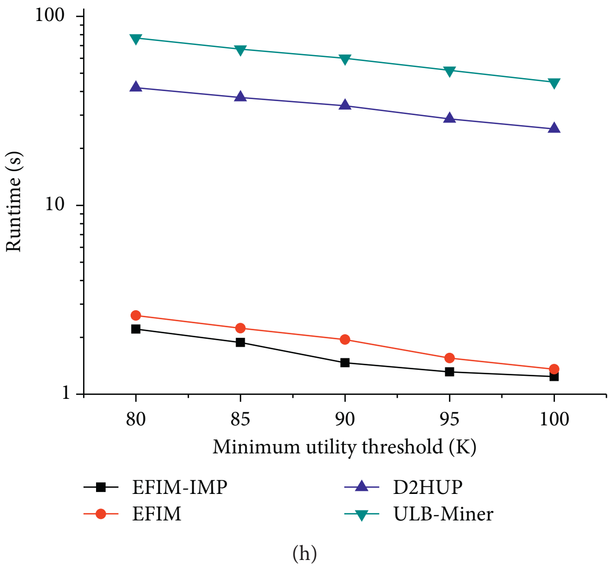
(e)



(f)

Figure 1: Continued.





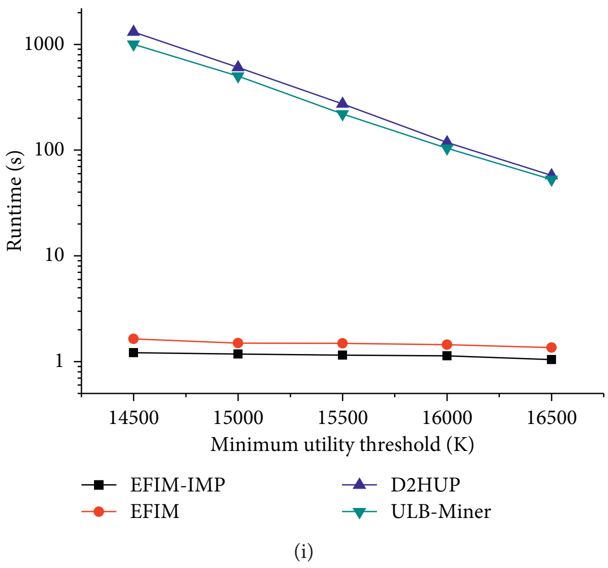


Figure 1: Running time on different datasets. (a) Accident. (b) BMS. (c) Foodmart. (d) Chainstore. (e) Kosarak. (f) Pumsb. (g) Chess. (h) Mushroom. (i) Connect.

local dataset, denoted as β-D) and generates two lists *Primary* (β) and *Secondary* (β), as the local candidate lists.

The global candidate lists generated by EFIM in Algorithm 1 may contain noncandidate items; so do the local candidate lists generated by Algorithm 2: they may also contain noncandidates and hence reduce the mining efficiency. So, we propose Strategy 2 and apply it to the local dataset processing of EFIM; the improved algorithm is shown in Algorithm 4.

Strategy 2. Repeatedly calculate *TWU* values and delete noncandidate items from the local header table, till the recalculation does not generate new noncandidate items.

Strategy 2 is augmented to the local data processing of EFIM (between Line 7 and Line 8 in Algorithm 2); the revised procedure (named Search-IMP) is shown in Algorithm 4: Lines 9–10 record the number of candidates before and after recalculation, respectively, to *count0* and *count1*; if the two counts differ, indicating that the local candidates have been changed, the proposed strategy is applied to delete those newly established noncandidates (Line 12), recalculate lists *Secondary* (β) and *Primary* (β) (Lines 13–15), recount items in these lists, and repeat the above process if the counts differ (Lines 16–17, Line 11), till *Secondary* (β) is stable through this process.

Algorithm 2 needs an additional scan on dataset α-D when processing each item to obtain subdataset β-D; to optimize this step, Algorithm 4 maintains an index of candidate items in each transaction (line 1) of α-D, to enable fast searches for transactions containing a certain item and locating the position of this item (line 4).

```
     Input: $D$: a transaction database; $MinU$: a user-specified threshold.
     Output: the set of high-utility itemsets
 (1) $\alpha = \varnothing$
 (2) Calculate $lu\ (\alpha,i)$ for all items $i \in I$ by scanning $D$, using a utility-binary;
 (3) $Secondary\ (\alpha) = \{i | i \in I \land lu\ (\alpha,i) \geq MinU\}$;
 (4) $count0$ = number of items in $I$;
 (5) $count1$ = number of items in $Secondary\ (\alpha)$;
 (6) while $count0 - count1 > 0$ do
 (7)     Remove each item $i \notin Secondary\ (\alpha)$ from the transactions;
 (8)     Recalculate $lu(\alpha,i)$ for all items $i \in I$ by scanning $D$;
 (9)     $Secondary\ (\alpha) = \{i | i \in I \land lu(\alpha,i) \geq MinU\}$;
(10)     $count0 = count1$;
(11)     $count1$ = number of items in $Secondary\ (\alpha)$;
(12) end while
(13) Let $\succ$ be the total order of TWU ascending values on $Secondary\ (\alpha)$;
(14) Scan $D$ to remove each item $i \notin Secondary\ (\alpha)$ from the transactions, sort items in each transaction according to $\succ$, and delete
     empty transactions;
(15) Sort transactions in $D$ according to $\succ T$;
(16) Calculate the subtree utility $su(\alpha,i)$ of each item $i \in Secondary\ (\alpha)$ by scanning $D$, using a utility-bin array;
(17) $Primary(\alpha) = \{i | i \in Secondary\ (\alpha) \land su(\alpha,i) \geq MinU\}$;
(18) Search ($\alpha$-$D$, $Primary(\alpha)$, $Secondary\ (\alpha)$, $MinU$);
```

Algorithm 3: The EFIM-IMP algorithm.

3.3. Algorithm Analysis. Both our two improved strategies adopt the approach of deleting newly established noncandidates to reduce the number of candidate items in the global/local header table. The criteria for screening noncandidates are based on whether the *TWU* value of an item is less than the minimum threshold; and, according to Property 1, these strategies will not affect the mining result; that is, they do not cause loss on high-utility patterns during the mining process.

The time complexity of EFIM is $O\ (lnw)$, where l is the number of candidate items, n is the number of total transactions in the dataset, and w is the average length of transactions. The time complexity of the augmented part of our proposed strategy is $O\ (krw)$, where k is the number of loops, r is the number of transactions containing noncandidates, and w is the average length of transactions.

The core functionality of our improved strategy is to delete newly discriminated noncandidates constantly, that is, to reduce the number of candidates (l), the average length of transactions (w), and even the total number of transactions (n); so, the more noncandidates it deletes, the more efficiency boost will be achieved. The time complexity of the revised algorithm EFIM-IMP is still $O(l_1 n_1 w)$, where l_1 is the number of candidate items, n_1 is the average number of transactions containing candidate items in the dataset, and w is the average length of transactions. According to strategies 1 and 2, l_1 is not bigger than l, and n_1 is not bigger than n. So, $O(l_1 n_1 w)$ is not bigger than $O\ (lnw)$.

But if there are not so many noncandidates to purge, due to the time-costing characteristics of the strategy itself, the overall performance boost for the mining process might be hampered, even be reduced to less efficient than the original EFIM.

4. Experiments

The improved algorithm of EFIM (we call EFIM-IMP hereafter) is the integration of EFIM with our proposed strategies. We compared the performance of EFIM-IMP with three algorithms, EFIM, D2HUP, and ULB-Miner. Source code of EFIM, D2HUP, and ULB-Miner can be downloaded from the website, http://www.philippe-fournier-viger.com/spmf/, and EFIM-IMP is a direct revise upon the source of EFIM. All programs are written in Java. Experimental platform, Windows 7 operating system, 16G of memory, Intel (*R*) Core I i7-6500 CPU @ 2.50 GHz, and Java Heap space, is set to 1.5 G. Experimental comparisons of these four algorithms are carried out on classic datasets. Nine standard datasets were used for our experiments, including 2 high volume dataset (*Chain-store* and *Kosarak*). These datasets can also be downloaded from the above website. Table 3 shows the characteristics of these nine datasets.

We ran four algorithms on different datasets while decreasing the minimum utility threshold. On the dataset Pumsb, the algorithms D2HUP and ULB-Miner were slow, for example, D2HUP ran 930 s and ULB-Miner ran 650 s, so we did not run these two algorithms on the dataset Pumsb under different thresholds. Figure 1 is the time cost comparison of the four algorithms on different datasets. Figure 2 is the memory cost comparison of the four algorithms on different datasets. Multiple runs are conducted, and numbers are recorded and averaged as the final experimental results.

We can see from Figure 1 that the revised algorithm EFIM-IMP outperforms on running time except for the dataset Foodmart. EFIM-IMP is faster than EFIM on each

```
    Input: $\alpha$: an itemset; $\alpha$-$D$: the $\alpha$ projected database; $Primary(\alpha)$: the primary items of $\alpha$; $Secondary(\alpha)$: the secondary items of $\alpha$;
    $MinU$: a user-specified threshold.
    Output: the set of high-utility itemsets that are extensions of $\alpha$
 (1) Record transaction ID and item index of each item to two lists (TransList and ItemList) by scanning
 (2) for each item $i \in Primary(\alpha)$ do
 (3)     $\beta = \alpha \cup \{i\}$;
 (4)     Scan $\alpha - D$ to calculate $u(\beta)$ and create $\beta - D$//uses transaction merging;
 (5)     if $u(\beta) \geq minutil$ then output $\beta$;
 (6)     Calculate $su(\beta,z)$ and $lu(\beta,z)$ for all item $z \in Secondary(\alpha)$ by scanning $\beta - D$ once, using two utility-bin arrays;
 (7)     $Primary(\beta) = \{z \in Secondary(\alpha) | su(\beta,z) \geq MinU\}$;
 (8)     $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta,z) \geq MinU\}$;
 (9)     $count0$ = number of items in $Secondary(\alpha)$;
(10)     $count1$ = number of items in $Secondary(\beta)$;
(11)     while $count0 - count1 > 0$ do
(12)         Remove each item $z \notin Secondary(\beta)$ from the transactions in $\beta - D$;
(13)         Calculate $su(\beta,z)$ and $lu(\beta,z)$ for all items $z \in Secondary(\beta)$ by scanning $\beta - D$;
(14)         $Secondary(\beta) = \{z | z \in Secondary(\beta) \wedge lu(\beta,z) \geq MinU\}$;
(15)         Primary($\beta$) = $\{z | z \in Secondary(\beta) \wedge su(\beta,z) \geq MinU\}$;
(16)         $count0 = count1$;
(17)         $count1$ = number of items in $Secondary(\beta)$;
(18)     end while
(19)     Search($\beta$, $\beta - D$, $Primary(\beta)$, $Secondary(\beta)$, $MinU$);
(20) end for
```

Algorithm 4: The Search-IMP procedure.

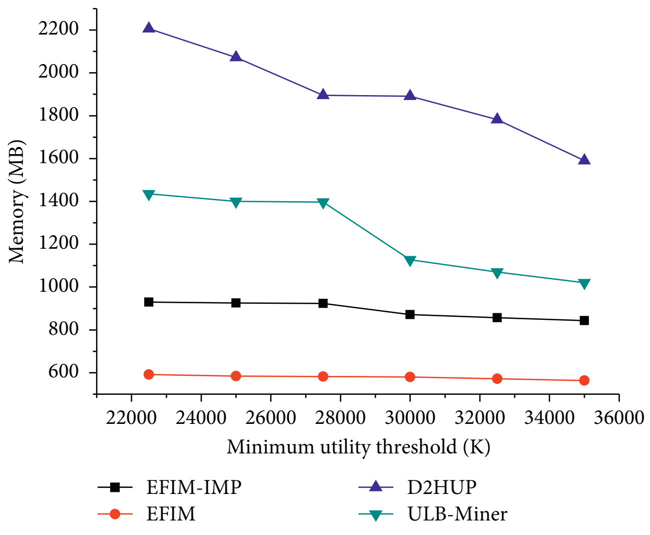
Table 3: Dataset characteristics.

Dataset	#Transactions	#Distinct items	Avg. trans. length	Type
Accident	340,183	468	33.8	Moderately dense, moderately long transactions
BMS	59,601	497	4.8	Sparse, short transactions
Foodmart	4,141	1559	4.4	Sparse, short transactions
Chainstore	1,112,949	46,086	7.2	Very sparse, short transactions
Kosarak	990,000	41,270	8.1	Very sparse, moderately short transactions
Pumsb	49,046	2,113	74	Dense, very long transactions
Chess	3,196	75	37	Dense, long transactions
Mushroom	8,124	119	23	Dense, moderately long transactions
Connect	67,557	129	43	Dense, long transactions

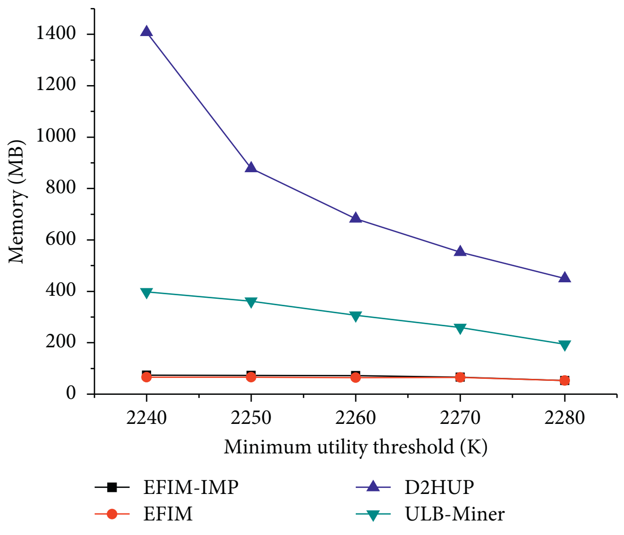
dataset. The revised algorithm EFIM-IMP has an obvious improvement on the datasets with more distinct items, for example, Chainstore and Kosarak. On the dense datasets, EFIM-IMP came near to D2HUP, but D2HUP used more memory than EFIM-IMP, as shown in Figure 2.

EFIM only calculates *TWU* values once, deletes non-candidate items according to the aforesaid values, and leaves all remaining items as candidates; EFIM-IMP iteratively repeats the *TWU* calculating and noncandidates deleting processes, till the number of candidates remains unchanged.

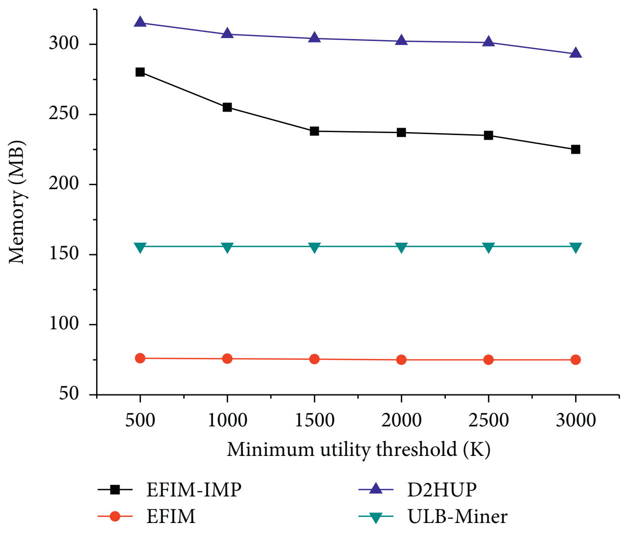
Among the aforementioned datasets, five of them (Accident, BMS, Chess, Connect, and Mushroom) include fewer distinct items (e.g., 75 distinct items in Chess), among which 4 datasets (except BMS) are dense. As a result, there are not so much number of candidate items to be cut down by strategies 1 and 2, and the improvement of algorithm EFIM-IMP is not obvious. As a contrast, for datasets Chainstore and Kosarak, the improvement of algorithm EFIM-IMP is obvious. The reason is that these two datasets include much more distinct items and transactions (e.g., Chainstore includes 46,086 distinct items and 1,112,949 transactions) and are sparse. Our strategies can efficiently reduce the number of candidate items on these two datasets.



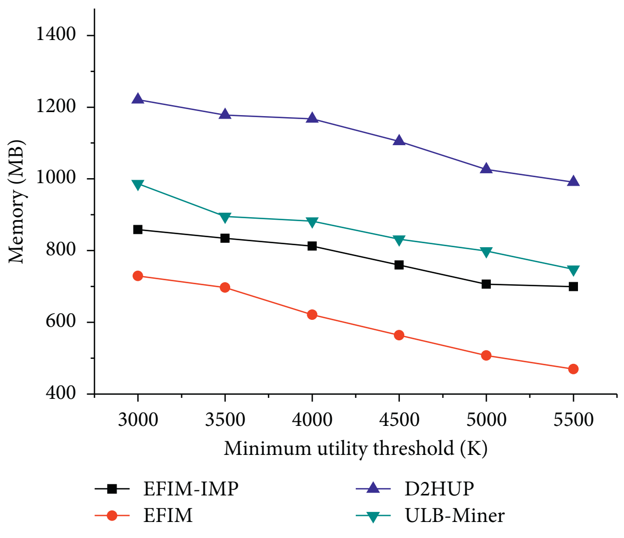
(a)



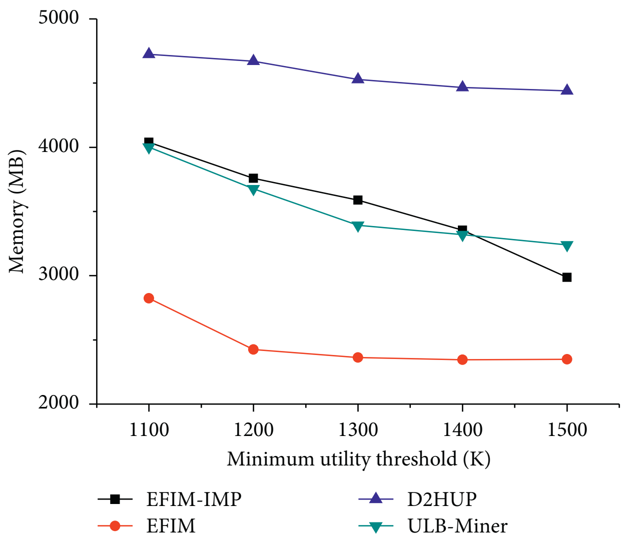
(b)



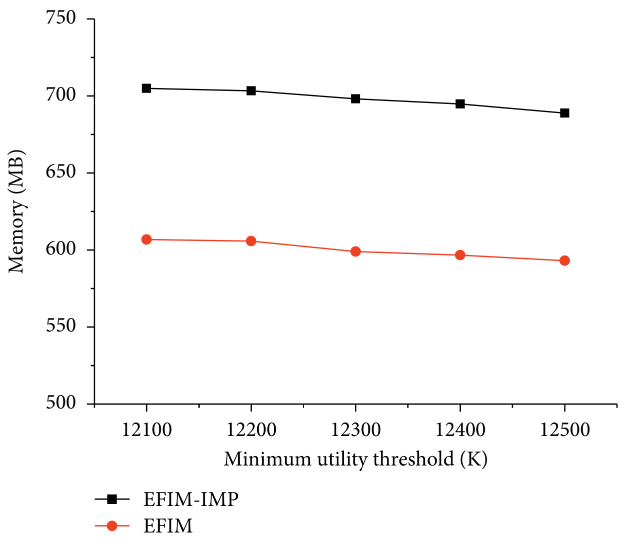
(c)



(d)

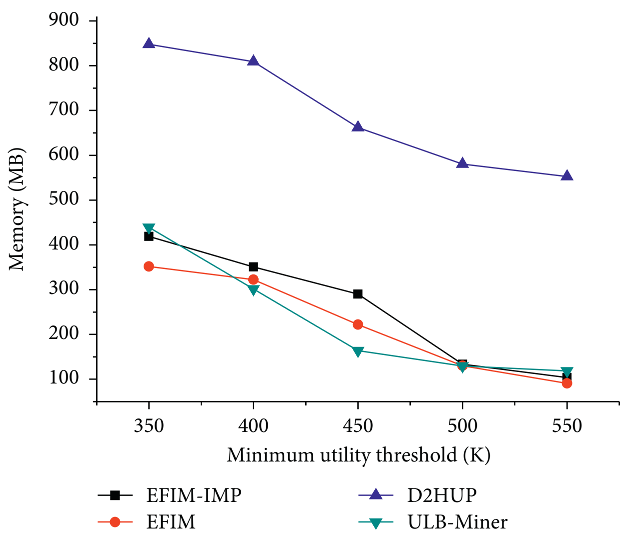


(e)

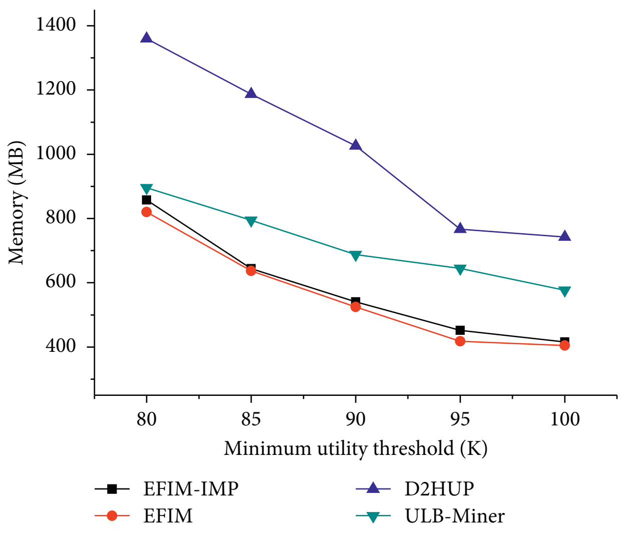


(f)

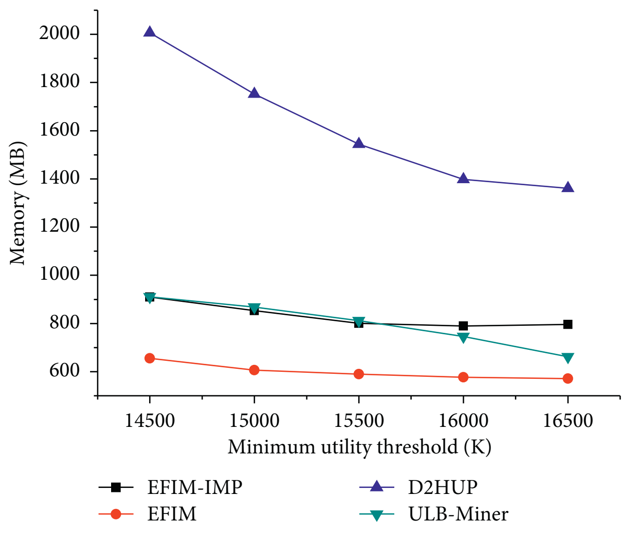
Figure 2: Continued.



(g)



(h)



(i)

Figure 2: Memory usage on different datasets. (a) Accident. (b) BMS. (c) Foodmart. (d) Chainstore. (e) Kosarak. (f) Pumsb. (g) Chess. (h) Mushroom. (i) Connect.

5. Conclusion

This paper focuses on optimization approaches of high utility pattern mining algorithms and proposes an improved strategy that, by iteratively removing newly discriminated noncandidate items, reduces the search space of the mining process and boosts mining efficiency. The proposed strategy was applied to algorithm EFIM. Nine standard datasets were used for algorithm verification, including 2 high volume datasets. Experimental results show that the improved algorithm can reduce the number of candidates effectively and outperform EFIM in time efficiency; the improvement is significant on high volume datasets.

Data Availability

The data used to support the findings of this study can be downloaded from the SPMF website (http://www.philippe-fournier-viger.com/spmf/).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the Zhejiang Basic Public Welfare Research Project (2017C35014), Zhejiang Philosophy and Social Science Project (19GXSZ49YB andd 21NDJC167YB), and the Ningbo Natural Science Foundation (2017A610122 and 2016A610042). The authors also want to express their sincere gratitude to Philippe Fournier-Viger and his SPMF website (http://www.philippe-fournier-viger.com/spmf/) for EFIM source code and test datasets.

References

[1] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1–12, Dallas, TX, USA, May 2000.

[2] C. F. Ahmed, S. K. Tanbeer, B. Byeong-Soo Jeong, and Y. Young-Koo Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708–1721, 2009.

[3] H. Li, H. Huang, Y. Chen, Y. Liu, and S. Lee, "Fast and memory efficient mining of high utility itemsets in data streams," in *Proceedings of Eighth IEEE International Conference on Data Mining*, pp. 881–886, IEEE, December 2008.

[4] B.-E. Shie, H.-F. Hsiao, and V. S. Tseng, "Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments," *Knowledge and Information Systems*, vol. 37, no. 2, pp. 363–387, 2013.

[5] A. Erwin, R. P. Gopalan, and N. R. Achuthan, "Efficient mining of high utility itemsets from large datasets," in *in Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 554–561, Springer, Osaka, Japan, May 2008.

[6] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data & Knowledge Engineering*, vol. 64, no. 1, pp. 198–217, 2008.

[7] M. Zihayat, H. Davoudi, and A. An, "Mining significant high utility gene regulation sequential patterns," *BMC Systems Biology*, vol. 11, p. 109, 2017.

[8] W. Gan, J. C. Lin, J. Zhang, P. Fournier-Viger, H. Chao, and S. Y. Philip, "Fast utility mining on sequence data," *IEEE Transactions on Cybernetics*, vol. PP, no. 89, 14 pages, 2020.

[9] J. C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, and V. S. Tseng, "Efficient algorithms for mining high-utility itemsets in uncertain databases," *Knowledge-Based Systems*, vol. 96, pp. 171–187, 2016.

[10] H. Yao, H. J. Hamilton, and G. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings of the 4th SIAM International Conference on Data Mining (ICDM 2004)*, pp. 482–486, Lake Buena Vista, FL, USA, April 2004.

[11] Y. Liu, W. K. Liao, and A. Choudhary, "A Two-phase algorithm for fast discovery of high utility itemsets," in *Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pp. 689–695, Hanoi, Vietnam, 2005.

[12] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems With Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[13] X. Han, X. Liu, J. Li, and H. Gao, "Efficient top-k high utility itemset mining on massive Data," *Information Sciences*, In Press, 2020.

[14] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772–1786, 2013.

[15] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems With Applications*, vol. 41, no. 8, pp. 3861–3878, 2014.

[16] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM 2012)*, pp. 55–64, Association for Computing Machinery, Maui, HI, USA, October 2012.

[17] J. Liu, K. Wang, and B. C. Fung, "Direct discovery of high utility itemsets without candidate generation," in *Proceedings of the IEEE 12th International Conference on Data Mining*, pp. 984–989, IEEE, Brussels, Belgium, December 2012.

[18] P. Fournier-Viger, C. Wu, S. Zida, and V. S. Tseng, "FHM: faster high-utility itemset mining using estimated utility Co-occurrence pruning," *Foundations of Intelligent Systems*, vol. 8502, pp. 83–92, 2014.

[19] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "EFIM: a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 595–625, 2017.

[20] S. Krishnamoorthy, "HMiner: efficiently mining high utility itemsets," *Expert Systems with Applications*, vol. 90, pp. 168–183, 2017.

[21] Q.-H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørvåg, and T.-L. Dam, "Efficient high utility itemset mining using buffered utility-lists," *Applied Intelligence*, vol. 48, no. 7, pp. 1859–1877, 2018.