Applied Intelligence (2023) 53:6992–7006
https://doi.org/10.1007/s10489-022-03850-4

An efficient utility-list based high-utility itemset mining algorithm

Zaihe Cheng[1,2] · Wei Fang[2] · Wei Shen[2] · Jerry Chun-Wei Lin[3] · Bo Yuan[4]

Accepted: 3 June 2022 / Published online: 13 July 2022

Abstract

High-utility itemset mining (HUIM) is an important task in data mining that can retrieve more meaningful and useful patterns for decision-making. One-phase HUIM algorithms based on the utility-list structure have been shown to be the most efficient as they can mine high-utility itemsets (HUIs) without generating candidates. However, storing itemset information for the utility-list is time-consuming and memory consuming. To address this problem, we propose an efficient simplified utility-list-based HUIM algorithm (HUIM-SU). In the proposed HUIM-SU algorithm, the simplified utility-list is proposed to obtain all HUIs effectively and reduce memory usage in the depth-first search process. Based on the the simplified utility-list, repeated pruning according to the transaction-weighted utilisation (TWU) reduces the number of items. In addition, a construction tree and compressed storage are introduced to further reduce the search space and the memory usage. The extension utility and itemset TWU are then proposed to be the upper bounds, which reduce the search space considerably. Extensive experimental results on dense and sparse datasets indicate that the proposed HUIM-SU algorithm is highly efficient in terms of the number of candidates, memory usage, and execution time.

Keywords Data mining · Pattern mining · High-utility itemset mining · Simplified utility-list

1 Introduction

Frequent itemset mining (FIM) [1–3] is an important task in data mining. In FIM, sets of frequent itemsets can be mined when the occurrence frequencies are greater than a specified minimum confidence threshold or a minimum support threshold [4]. However, FIM is inefficient for mining high-profit itemsets since only occurrence frequencies are considered for the itemsets. High-utility itemset mining (HUIM) is therefore proposed, where both frequent and profitable itemsets are taken into consideration [5–7]. In [8], HUIM is classified in a new mining framework, which is called utility-oriented pattern mining. HUIM has been used in a variety of real-world applications, including customer purchase trends in the retail market [9], customer segmentation [10], web service [11], and biomedical applications [12]. Many other important data mining tasks have also been studied that are inspired by HUIM, such as high-utility sequential pattern mining [13, 14], closed HUIM [15, 16], high-utility stream mining [17], top-K high-utility mining [18–21], local and peak HUIM [22], High average utility sequence mining [23, 24], periodic pattern mining [25], and HUIs with negative unit profits [26].

HUIM algorithms that are based on a utility-list, such as the HUI-Miner [27], the HUP-Miner [28], the FHM [29], the ULB-Miner [30], and the HMiner [31], have proven to be more efficient than the other algorithms [30] since the utility-list structure facilitates the direct calculation of the utility of itemsets without scanning the database. However, creating and maintaining utility-lists may lead to increased memory usage and time costs, especially on dense databases with long transactions. It is necessary to design more efficient algorithms to address the challenges in terms of memory and runtime consumption, both on dense datasets and sparse datasets. In this paper, we propose

✉ Wei Fang
fangwei@jiangnan.edu.cn

1 School of Internet of Things, Wuxi Institute of Technology, Gaolang Road, Wuxi, Jiangsu, 214121, China

2 Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Jiangnan University, Lihu Avenue, Wuxi, Jiangsu, 214122, China

3 Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

4 Computer Science and Engineering Department, Southern University of Science and Technology, Shenzhen, China

an efficient utility-list-based HUIM algorithm with the reorganised transition database (HUIM-SU) to address this problem. The major contributions of this paper are given below:

- A simplified utility-list is designed in the HUIM-SU algorithm, where each record represents all the utilities of the transactions related to an individual item.
- A construction tree is proposed to reduce the search space based on the simplified utility-list, and then compressed storage is introduced to reduce the memory usage of the construction tree.
- To further reduce the search space of the promising candidates, extension utility and local transaction-weighted utilisation (TWU) utility are employed as the upper bounds in the proposed HUIM-SU.
- With the purpose of greatly improving the computational efficiency for the extension utility and the itemset TWU utility greatly, the methods of the ordered datasets and the avoidance of repeated calculation are designed.

The rest of this paper is organised as follows: Related work is described in Section 2. Section 3 provides the problem definition of mining HUIs. The proposed HUIM-SU algorithm is presented in Section 4. In Section 5, we report the comparative experiments and the results. Conclusions are drawn in Section 6.

2 Related work

In HUIM, the superset of an itemset is neither anti-monotonic nor monotonic since the utility of the superset has more than one relationship than the utility of the itemset [32]. Thus, the utility cannot be used as the measurement to reduce the search space. A high utility superset may exist in the itemset with low utility, and if an algorithm ignores this, the discovered HUIs are incomplete. To solve this problem, several HUIM algorithms have been proposed to limit the search space by using the upper bounds of each itemset. The TWU concept was first introduced in the two-phase algorithm [33], and it can be used to ensure whether the itemset supersets have HUIs. By considering the pruning property of the TWU measure, all itemsets containing the itemset were lower than the threshold when the TWU of an itemset was lower than the threshold, $minutil$. TWU has been widely used in the HUIM algorithms, such as in EFIM [34], UP-Growth+ [12], ULB-Miner [30], PTM [7], NPHUIs [22], and HUI-MMU [35].

HUIM algorithms can be divided into two-phase algorithms and one-phase algorithms based on whether candidate HUIs are generated [5]. In two-phase algorithms, a set of candidate HUIs is generated in the first phase by overestimating their utilities. Then, the overestimated itemsets in the database are scanned and filtered according to a user-specified threshold in the second phase. The two-phase technique usually produces a great number of candidates, which leads to time-consuming database scans during the second phase.

To avoid the problem in two-phase HUIM algorithms, the utility of itemsets was proposed to be calculated in memory without generating candidate HUIs and scanning the database frequently for one-phase HUIM algorithms. To limit the search space in one-phase HUIM algorithms, upper bounds and pruning strategies were also used. HUI-Miner [27] was one of the earliest one-phase HUIM algorithms. The utility of the generated itemsets was calculated directly in HUI-Miner based on the utility-list structure, which could reduce the execution time and the memory usage and has been extensively used in the other one-phase HUIM algorithms. mHUIMiner [36], HUP-Miner [28] and FHM [29] are the improvements of HUI-Miner. In HUP-Miner, a partitioned utility-list structure and two pruning strategies were employed to improve performance. A new pruning strategy, i.e., EUCP, was proposed in FHM, which was much faster than HUI-Miner [29]. DHUP-Miner and its parallel version, DHUP-Miner*, have been studied in [37], with novel pruning strategies to reduce the search space. In UFH [38], a hybrid framework was used by combining the tree-based and the utility-list-based algorithms to mine HUIs efficiently. The utility-list structure and the improved versions have been used extensively in these algorithms to store the utility values of itemsets to mine HUIs efficiently. However, storing itemset information using the utility-list structure was time-consuming and memory-consuming, especially on dense databases with long transactions [30, 34].

3 Problem statement

A collection of transactions consists of a transaction database $D = \{T_1, T_2, \cdots, T_m\}$. Each transaction belonging to the database D has a unique identifier T_{ID}. $I = \{i_1, i_2, \cdots, i_n\}$ is a finite set of nonrepeating n items from D. For each transaction T, the itemset $X \subseteq I$ is a finite set of items. The internal utility (e.g., purchase quantity)and the external utility (e.g., unit profit) are both positive numbers that associated with each item. A sample transaction database is shown in Table 1 that contains five transactions [34] and the database is used as the running example. Transaction T_3 contains the items, b, c, d, and e with the internal utilities of 4, 3, 3, and 1, respectively. The external utilities of all the seven items are given in Table 2.

Table 1 Example transaction database

T_{ID}	Transaction
T_1	$(a, 1)(c, 1)(d, 1)$
T_2	$(a, 2)(c, 6)(e, 2)(g, 5)$
T_3	$(b, 4)(c, 3)(d, 3)(e, 1)$
T_4	$(a, 1)(b, 2)(c, 1)(d, 6)(e, 1)(f, 5)$
T_5	$(b, 2)(c, 2)(e, 1)(g, 2)$

Definition 1 (Utility of an item/itemset) Let there be a transaction T_j, an item i, and an itemset X. The utility of i in T_j is denoted as $u(i, T_j)$ and is calculated as $p(i) \times q(i, T_j)$, where $p(i)$ and $q(i, T_j)$ are the external utility and the internal utility of the item respectively. The utility of X in T_j is defined as $u(X, T_j) = \sum_{i \in X} u(i, T_j)$. The utility of X is defined as $u(X) = \sum_{T_j \in g(X)} u(X, T_j)$, where $g(X)$ is the set of transactions containing X.

For instance, the utility of item b in T_4 is $u(b, T_4) = 2 \times 2 = 4$, and the utility of the itemset $\{a, c\}$ in T_1 is $u(\{a, c\}, T_1) = u(a, T_1) + u(c, T_1) = 5 \times 1 + 1 \times 1 = 6$. The utility of the itemset $\{c, d\}$ is $u(\{c, d\}) = u(\{c, d\}, T_1) + u(\{c, d\}, T_3) + u(\{c, d\}, T_4) = u(c, T_1) + u(d, T_1) + u(c, T_3) + u(d, T_3) + u(c, T_4) + u(d, T_4) = 1 + 2 + 3 + 6 + 1 + 12 = 25$.

The computational complexity of computing $u(X)$ is $O(m * n)$ in general. For some HUIM algorithms, such as EFIM, a binary search technique is used to improve the search efficiency for X, and then the complexity is $O(m * log_2 n)$.

Definition 2 (High-utility itemset (HUI)) Let $minutil$ be a user-specified threshold with a positive value. If the utility $u(X)$ is no less than $minutil$, then X is a high-utility itemset (HUI); otherwise, it is a low-utility itemset.

For instance, if $minutil = 32$, the HUIs in the sample database are $\{b, c, d\}$, $\{b, d, e\}$, and $\{b, c, d, e\}$ with the utilities of 34, 36, and 40, respectively.

Definition 3 (Transaction utility (TU) and TWU) Let T_j be a transaction and x be an item. The transaction utility of T_j is $TU(T_j) = \sum_{x \in T_j} u(x, T_j)$. The TWU of x is $TWU(x) = \sum_{T_j \in g(x)} TU(T_j)$.

For instance, the TWU of item a is $TWU[a] = TU[T_1] + TU[T_2] + TU[T_3] = 5 + 1 + 2 + 10 + 6 + 6 + 5 + 5 + 4 + 1 + 12 + 3 + 5 = 65$. In the sample transaction database, the TWUs of all the items are shown in Table 3.

Property 1 (Pruning the search space using the TWU) itemset X and all its supersets are low-utility itemsets if $TWU(X)$ is less than $minutil$ [33].

Definition 4 (Remaining utility) Let there be an itemset X, a transaction T_j, and a total order $\succ$ on the items from I. The set of all the items after X in T_j is denoted as $\{i \in T_j \wedge (i \succ x, \forall x \in X)\}$. The remaining utility of X in T_j is $re(X, T_j) = \sum_{i \in T_j \wedge (i \succ x, \forall x \in X))} u(i, T_j)$ regarding the sum of the utilities for the set $\{i \in T_j \wedge (i \succ x, \forall x \in X)\}$.

For example, the remaining utility of itemset $\{a, c\}$ in T_4 is $re(\{a, c\}, T_4) = u(d, T_4) + u(e, T_4) + u(f, T_4) = 12 + 3 + 5 = 20$.

Definition 5 (Utility-list) Let there be an itemset X and a transaction T_j containing X. The utility-list of X contains a set of tuples (T_j, iutil, rutil) for each T_j. Here, $iutil = u(X, T_j))$ and rutil = $(re(X, T_j))$.

The utility-list of $\{a, c\}$ in the sample database is $\{(T_1, 6, 2), (T_2, 16, 11), (T_4, 6, 20)\}$.

Definition 6 (Remaining utility upper bound) Let there be an itemset X and an item i. The extension of X can be obtained by appending i to X, which satisfies $i \succ x, \forall x \in X$. The remaining utility upper bound of X is $reu(X) = u(X) + re(X)$.

For example, the remaining utility upper-bound of $\{a, c\}$ is $reu(\{a, c\}) = u(\{a, c\}) + re(\{a, c\}) = 6 + 2 + 16 + 11 + 6 + 20 = 61$

Property 2 (Pruning search space using utility-lists) Let X be an itemset. If $reu(X) < minutil$, then X and all of its extensions are low-utility itemsets, which can be pruned in the search space [27].

4 The proposed HUIM-SU algorithm

The proposed HUIM-SU algorithm is a one-phase HUIM algorithm that introduces some ideas to mine HUIs efficiently in terms of memory usage and execution time

Table 2 External utility values

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Table 3 TWU of each item

Item	a	b	c	d	e	f	g
TWU	65	61	96	58	88	30	38

on both sparse and dense datasets. To reduce the number of database scans and improve the calculation efficiency of the utility of an item or an itemset, the utility-list is simplified at first. A repeated pruning strategy based on TWU is then performed on the simplified utility-list with the purpose of reducing the number of items. The construction tree is introduced to reduce the search space with the proposed compressed storage to reduce the memory usage. By introducing the extension utility and the itemset TWU as upper bounds, the width and the depth of the search space can be reduced. The ordered items have helped to avoid the repeated calculation for the extension utility.

4.1 Simplified utility-list

According to Definition 5., the utility-list is a set of tuples in the form (T_j, iutil, rutil). In this paper, the simplified utility-list is a set of tuples in the form (T_j, iutil). The iutil is the utility of an itemset in T_j and is defined as $u(i, T_j)$. With the simplified utility-list, calculating the utility only requires scanning one-or-two item data (ITD) rather than scanning the entire database or utility-list structure. Then, the utility of an item and the sum of the utilities of two items can be calculated with linear time complexity.

Definition 7 (Item Data (ITD)) Let there be an item i and a transaction T_j. The ITD is a set of records $(T_j, u(i, T_j))$, where $u(i, T_j)$ is the utility of i in T_j.

According to the ITDs in Table 4, the utility of item a is calculated as $u(a) = u(a, T_1) + u(a, T_2) + u(a, T_4) = 5+10+5 = 20$. The utility of itemset $\{a, c\}$ is calculated as $u(\{a, c\}) = u(\{a, c\}, T_1) + u(\{a, c\}, T_2) + u(\{a, c\}, T_4) = u(a, T_1) + u(c, T_1) + u(a, T_2) + u(c, T_2) + u(a, T_4) + u(c, T_4) = 5 + 1 + 10 + 6 + 5 + 1 = 28$.

4.2 Pruning search space repeatedly based on TWU

Property 1 is very useful for reducing the search space and has been used in several HUIM algorithms. For any item or itemset α, if TWU(α) $< minutil$, α and its supersets are all low utility and should be deleted. Based on Property 1, we propose an improved pruning strategy that can prune the search space repeatedly according to TWUs until the TWUs of all items are greater than the $minutil$. The TWUs of the remaining items are updated once an item is deleted. The time complexity of updating TWUs is $O(n^3)$ on the original transaction database. With the proposed RTD, updating the TWUs can be realized with the time complexity, $O(n^2)$.

Definition 8 (Updating TWU) Let i be a deleted item. The transactions in $ITD(i)$ are scanned to update the TWUs of all the items. $TWU[x] = TWU[x] - u(i, T_j), x \in T_j \in ITD(i)$.

For example, we consider the item f in Table 3 and $minutil = 32$, while the TWU of f is 30, which should be deleted. The ITD of f is $(T_4, 5)$ and $T_4 = \{(a, 1), (b, 2), (c, 1), (d, 6), (e, 1), (f, 5)\}$. Updating the TWUs of the items in T_4 after deleting item f are calculated as $TWU[a] = TWU[a] - u(f, T_4) = 65 - 5 = 60, TWU[b] = TWU[b] - u(f, T_4) = 61 - 5 = 56, TWU[c] = TWU[c] - u(f, T_4) = 96 - 5 = 91, TWU[d] = TWU[d] - u(f, T_4) = 58 - 5 = 53, TWU[e] = TWU[e] - u(f, T_4) = 61 - 5 = 56, TWU[f] = TWU[f] - u(f, T_4) = 30 - 5 = 25$.

4.3 The construction tree with compressed storage

Let X be an itemset with k items sorted in order $\succ$ of TWU. Obviously, the search space grows exponentially with the increasing k. To address this issue, we propose the construction tree to reduce the search space.

Definition 9 (Construction tree) A construction tree is built by traversing the transaction database, and the nodes are added according to the sorted items based on their TWUs. In the construction tree, the items with lower TWUs are regarded as the child nodes of the related item. The construction process is completed until all the transactions are visited.

According to the TWUs in Table 3, the order of the items is $\{f, g, d, b, a, e, c\}$. Initially, the root of the construction tree is empty. After scanning T_1, items d, a, and c are added as the nodes in the construction tree in order as shown in Fig. 1. Since the TWUs of items a and c are less than d, they should also be in the subtree of item d. Figure 2 shows the tree after scanning T_2. In addition, Fig. 3 shows the tree after scanning T_3. After scanning T_4, the overall construction tree can be obtained. To reduce the memory consumption of the construction tree, we propose the compressed storage to store items with the same parent

Table 4 Sample reorganized transaction database

Items	ITDs
a	$(T_1, 5)(T_2, 10)(T_4, 5)$
b	$(T_3, 8)(T_4, 4)(T_5, 4)$
c	$(T_1, 1)(T_2, 6)(T_3, 3)(T_4, 1)(T_5, 2)$
d	$(T_1, 2)(T_3, 6)(T_4, 12)$
e	$(T_2, 6)(T_3, 3)(T_4, 3)(T_5, 3)$
f	$(T_4, 5)$
g	$(T_2, 5)(T_5, 2)$

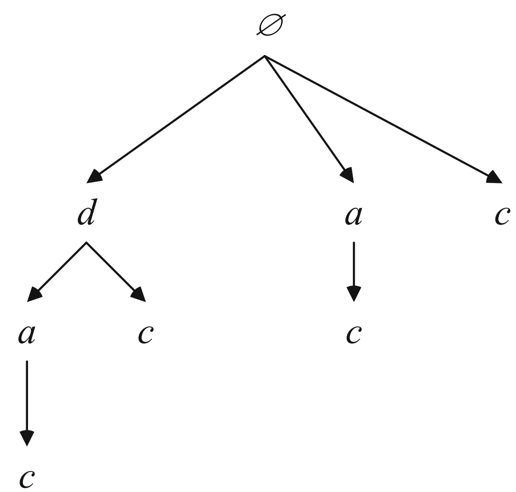


Fig. 1 Construction tree after scanning T_1

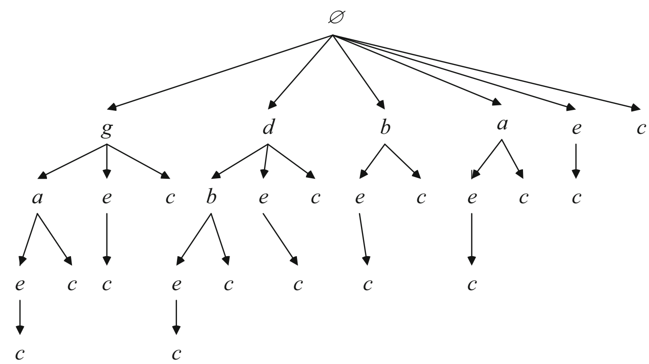


Fig. 3 The construction tree after scanning T_1, T_2, and T_3

item in the construction tree. The extension of an item or an itemset (EOI) is introduced to realise compressed storage.

Definition 10 (Extension of an item/ itemset (EOI)) The extension of an item (EOI) i is the set of the items behind i according to the order $\succ$ by scanning transactions containing i, as $EOI(i) = \{c \| c \in T_j \cap i \succ c\}$. The extension of an itemset $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_k\}$ is defined as the intersection of EOIs of all the items in the itemset, as, $EOI(\alpha) = EOI(\alpha_1) \cap EOI(\alpha_2) \cap \cdots \cap EOI(\alpha_k)$. The EOI is also sorted in the order $\succ$.

For example, EOI(g) = $\{a, e, c\}$. Compared with the itemset behind item g, which is $\{d, b, a, e, c\}$, the number of items in EOI has been reduced. According to the TWUs in Table 3, EOI(g) has fewer items, which makes it clear that the search space has been reduced. Given the itemset $\{b, e\}$, $EOI(\{b, e\}) = EOI(b) \cap EOI(e) = \{a, e, c\} \cap \{a, c\} = \{a, c\}$, the EOIs of the items in the sample transaction database are listed in Table 5. We can find that the EOIs in Table 5 can be stored in a compressed form, which also saves the structural information of the construction tree. As the items in EOI are in order, the EOI of an itemset can be calculated in linear time complexity.

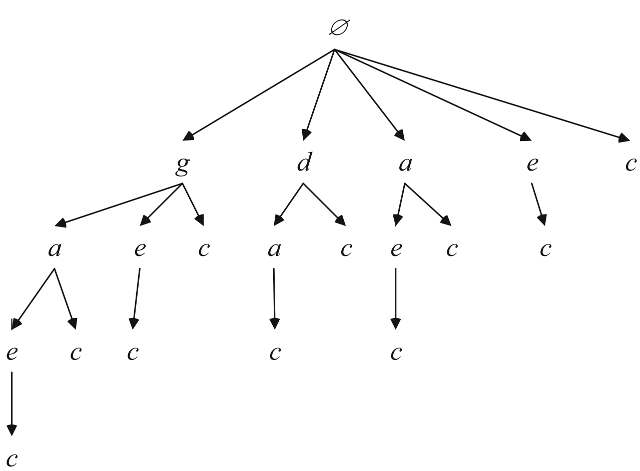


Fig. 2 Construction tree after scanning T_1 and T_2

There are two reasons why the sorted order of TWU is used in the proposed algorithm. First, according to the sorted TWUs, the construction tree is usually smaller than that by the disordered TWU. Second, if the TWU of an item is larger than its child nodes in the construction tree, the extension utilities of child nodes are then larger, which results in the larger search space after pruning and the computational time of the algorithm is therefore increased. In contrast, the small TWU of an item is helpful for reducing the search space, and then decreases the computational time.

4.4 Pruning the search space by two upper bounds

This subsection discusses the key procedure of HUIM-SU for pruning the search space, where two upper bounds on the utility of itemsets named extension utility and itemset TWU are introduced. The previous subsections have introduced the techniques used in this procedure.

The extension utility of item i is the sum of its utility, the utility of its parent node j in the construction tree, and the utilities of the items in EOI(i, j). If the extension utility of item i is less than $minutil$, the subtree of i should be

Table 5 Extension of items

Item	EOI
f	d, b, a, e, c
g	a, e, c
d	b, a, e, c
b	a, e, c
a	e, c
e	c
c	

pruned. The itemset TWU is used to reduce the number of items in ITD(j). If item c belongs to EOI(j), the itemset TWU of c is the sum of TWU of the transactions containing j and c .

Definition 11 (Extension utility) Let X be an itemset and an item $i \in EOI(X)$. The extension utility of i w.r.t. X is $eu(X, i) = \sum_{T_j \in EOI(X) \cap EOI(i)} (u(i, T_j) + u(X, T_j) + \sum_{x \in EOI(X,i)} u(x, T_j))$.

For example, let X be $\{d, b\}$. Then, its ITD is $\{(T_3, 14)(T_4, 16)\}$ and its EOI is $\{a, e, c\}$. The extension utility of $\{X, e\}$ is $eu(X, e) = u(\{d, b\}, T_4) + u(e, T_4) + u(c, T_4) + u(\{d, b\}, T_3) + (e, T_3) + u(c, T_3) = 16 + 3 + 1 + 14 + 3 + 3 = 40$.

Definition 12 (Pruning using extension utility) Let X be an itemset. If $eu(\{X, i\}) < minutil, i \in EOI(X)$, then the utilities of $\{X, i\}$ and their supersets are less than $minutil$. In other words, the subtree can be subtracted.

Definition 13 (Local TU) Let X be an itemset and a transaction $T_j \in ITD(X)$. The local TU of T_j is defined as $lTU(T_j) = u(X, T_j) + \sum_{i \in EOI(X)} u(i, T_j)$.

For example, let X be $\{d, b\}$, its ITD is $\{(T_3, 14)(T_4, 16)\}$, and its EOI is $\{a, e, c\}$. Then $lTU(T_4) = u(X, T_4) + u(a, T_4) + u(e, T_4) + u(c, T_4) = 16 + 5 + 3 + 1 = 25$.

Definition 14 (Itemset TWU (itwu)) The itemset TWU (itwu) is the sum of the TUs of the transactions that contain itemset X, which is defined as $itwu(X) = \sum_{T_j \in g(X)} TU(T_j)$.

For instance, the TWU of itemset $\{a, c\}$ is $itwu\{a, c\} = TU[T_1] + TU[T_2] = 5 + 1 + 2 + 10 + 6 + 6 + 5 = 35$.

Here, the difference between the extension utility upper bound and the itemset TWU upper bound is illustrated with an example. Given the node α with its subtree as shown in Fig. 4, then EOI(α) = $\{A, B, C, D\}$. The extension utility upper bound is first used. If node C in the second level with $eu(C) < minutil$, then the node C with its subtree is removed, which is marked red in Fig. 4. Then, the itemset TWU upper bound is used in the following step; if $lu(\alpha, C) < minutil$, then all the other subtrees with the root node C should be removed, which is marked blue in Fig. 4.

To realise the pruning procedure more clearly, we introduce the $widthnode$ and the $depthnode$ in the proposed HUIM-SU algorithm. The $widthnode$ is used to store the items in $EOI(X)$ containing HUIs and the $depthnode$ is used to determine the scope of the EOI of $EOI(X)$.

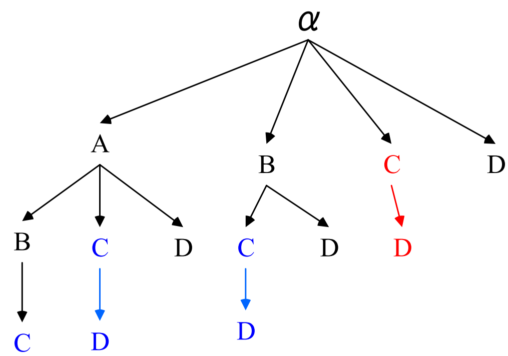


Fig. 4 Using extension utility to prune (red nodes) and itemset TWU to prune (blue nodes)

Definition 15 (Widthnode and Depthnode) Let X be an itemset. The widthnode of X is defined as $widthnode(X) = \{i \| i \in EOI(X) \wedge eu(X, i) \geq minutil\}$. The depthnode of X is defined as $depthnode(X) = \{i \| i \in EOI(X) \wedge itwu(X, i) \geq minutil\}$. Since $itwu(X, i) \geq eu(X, i)$, the relationship widthnode(X) $\subseteq$ depthnode(X) holds.

4.5 Transaction array for an extension utility and itemset TWU

The time complexity of calculating the extension utility is $O(l_i \cdot l_e)$, where l_i and l_e are the lengths of ITD and EOI, respectively. The previous extension utility contains the latter extension utility during the calculation process of all EOIs. Then the transaction array (TA) is introduced to store the utility of the EOI for fast calculation of the extension utility. The extension utility can be calculated from the last to the first according to the EOI order.

Definition 16 (Transaction array) For a transaction database D with a collection of m transactions, a transaction array is an array with a length of m, where each element in the array stores the utility of a transaction that is calculated based on ITD. By using a transaction array, the extension utility can be calculated easily and quickly.

For example, for itemset $\{b\}$ in Table 4, the initial TA is $\{(T_3, 8)\ (T_4, 4)\ (T_5, 4)\}$. By adding ITD($c$), ITD($e$), and ITD($a$) with the utilities in Table 4, the final TA is $(T_3, 14)$, $(T_4, 13)$, and $(T_5, 9)$. Table 6 shows the results of the extension utilities based on the TA.

The calculation process for itemset TWU is the same as that for the extension utility. Following the example in

Table 6 Extension utility calculation process

TA	T_3	T_4	T_5	Extension utility
Initialization	8	4	4	-
Add ITD(c)	11	5	6	22
Add ITD(e)	14	8	9	31
Add ITD(a)	-	13	-	13

Table 6, an illustration of the calculation for $itwu(b, c)$, $itwu(b, e)$, and $itwu(b, a)$ is given in Table 7. That is, $itwu(b, c) = TU[T_3]+TU[T_4]+TU[T_5] = 20+30+11 = 61$, $itwu(b, e) = TU[T_3]+TU[T_4]+TU[T_5] = 20+30+11 = 61$, $itwu(b, a) = TU[T_4] = 30 = 30$. Actually, the calculation process for itemset TWU is combined with the calculation process for the extension utility. Table 8 gives the TUs of items after removing item *f*.

4.6 The proposed HUIM-SU algorithm

The proposed HUIM-SU algorithm is introduced here with the ideas that were described in the previous subsections. The main procedure of the HUIM-SU (Algorithm 1) takes transaction database D and the $minutil$ as inputs. The initial itemset α is an empty set, the TWU of each item is calculated, and the simplified utility-list is constructed (line 1). Then, repeated pruning of the search space based on TWU is employed to reduce the number of items until the TWUs of the remaining items are all larger than $minutil$ (lines 2 to 8). The remaining items are sorted and the set with the sorted items is denoted as I^*. The EOIs of the items in I^* are calculated (line 9). The widthnode of each item in I^*, which is referred to as the rootwidthnode, is solved by comparing the extension utility with $minutil$ (line 10). A loop is performed to obtain the HUIs according to the depth-first search procedure from each rootwidthnode (lines 11 to 22). Each rootwidthnode is added to α (line 12). If the TWU of α is larger than the $minutil$, then α is output (lines 13 to 15). The widthnode of EOI (rootwidthnode) is then solved (line 16). For each widthnode, the depthnode is solved (line 18), and then the search procedure for the HUIs is performed (line 19). The last item is removed from α (line 21).

Algorithm 2. presents the depth-first search procedure for HUIs for each item of widthnode. The search procedure takes itemset α, widthnode, depthnode, the ITD of α, and $minutil$ as parameters. The procedure scans the items in widthnode to determine whether it is an HUI (lines 4 to 6). The extension utility upper bound is used to obtain the new widthnode (line 7). For the items in the new widthnode, the new depthnode is obtained by the itemset TWU upper bound (line 9). The search procedure is recursively called with a set of parameters to process the depth-first search (line 10).

Table 7 Itemset TWU calculation process

–	T_3	T_4	T_5	itwu
Add TWU(c)	20	30	11	61
Add TWU(e)	20	30	11	61
Add TWU(a)	–	30	–	30

It should be noted that an efficient projection-based indexing approach for the HUIM is proposed to reduce memory consumption and to speed up the mining process in [39], which shares a similar overall algorithm structure with the HUIM-SU. The main difference between these two algorithms lies in the data organisation and the pruning strategies. In the HUIM-SU, the simplified utility-list is designed, which is a set of tuples in the form (T_j, iutil), while an indexing structure is proposed with an index table using two fields of a transaction identifier and a last item position in [39]. For the pruning strategies, the search space is pruned twice based on TWU in the HUIM-SU, while a simple pruning technique based on the TWU model is appointed to reduce the number of unpromising candidate itemsets in [39].

Algorithm 1 Proposed HUIM-SU algorithm.

Input: D: a transaction database, $minutil$: the user-specified threshold;
Output: Set of HUIs;
1: $\alpha = \phi$, and find itemset I and calculate the TWU of each item, reorganize the data structure of D to D^*;
2: **while** $TWU[x] \geq minutil, \forall i \in I$ **do**
3: **for** $i = 0; i < I.length; i++$ **do**
4: **if** $TWU[I(i)] < minutil$ **then**
5: I.remove(i), update TWU of other items;
6: **end if**
7: **end for**
8: **end while**
9: Let I^* be the set of sorted items in I according to the TWUs. The EOIs of each item in I^* are calculated;
10: $rootWidthnode = \{z \| eu(z) \geq minutil, z \in I^*\}$;
11: **for** $i = 0; i < rootWidthnode.length; i++$ **do**
12: α.add(rootWidthnode[i]);
13: **if** TWU[α]$\geq minutil$ **then**
14: out(α);
15: **end if**
16: $Widthnode = \{z \| eu(z) \geq minutil, z \in EOI(rootWidthnode[k])\}$;
17: $Depthnode = \{z \| itwu(z) \geq minutil, z \in EOI(rootWidthnode[i])\}$;
18: Search(α, Widthnode, Depthnode, ITD[Widthnode[i]], $minutil$) (in Algorithm 2.);
19: α.removelast;
20: **end for**

Table 8 TUs of items after removing item f

Transaction	T_1	T_2	T_3	T_4	T_5
TU	8	27	20	28	11

Algorithm 2 Search algorithm.

Input: α: itemset; Widthnode; Depthnode; ITD(α); $minutil$;
Output: the set of HUIs;
1: **for** $i = 0; i < Widthnode.size; i++$ **do**
2: $item = Widthnode[i]$;
3: α.add(item);
4: **if** $TWU[\alpha] \geq minutil$ **then**
5: out(α);
6: **end if**
7: $newWidthnode = \{z \| eu(z) \geq minutil, z \in EOI(item) \cap Depthnode\}$;
8: $newDepthnode = \{z \| itwu(z) \geq minutil, z \in EOI(item) \cap Depthnode\}$;
9: Search(α, newWidthnode, newDepthnode, $ITD[item] \cap ITD(\alpha), minutil$);
10: α.removelast();
11: **end for**

4.7 A detailed example for HUIM-SU

In this subsection, we provide a comprehensive example to introduce how to carry out the proposed HUIM-SU. We consider the transaction database shown in Table 1 with the external utility values shown in Table 2 and $miniutl = 40$. The main procedure of the HUIM-SU (Algorithm 1.) is executed as follows.

Step 1 (line 1). Initially, itemset α is set to $\varnothing$. The transaction database D is scanned to calculate the TWU of all seven items, which is given in Table 3. The transaction database is reorganised with the resulting database D^* shown in Table 4.

Step 2 (lines 2 to 8). The search space is repeatedly pruned based on TWU until the remaining items have $TWU > minutil$. The first item to be removed is f since $TWU(f) = 30 < 40 = minutil$. Then, the TWUs of the other items are changed as shown in Table 9. The next removed item is g, and the updated TWUs are provided in Table 10 where all the items having $TWU > 40 = minutil$, and then the pruning process stops.

Step 3 (line 9). The items in Table 10 are sorted according to TWUs with the result itemset $I^* = \{d, b, a, e, c\}$. The RTD of I^* is shown in Table 11. Then, by scanning the original transaction database D, the EOI of each item in I^* is obtained as shown in Table 12.

Table 9 TWUs of items after removing item f

Items	a	b	c	d	e	g
TWU	60	56	91	53	83	38

Table 10 TWUs of items after removing item g

Items	a	b	c	d	e
TWU	55	54	84	53	76

Step 4 (line 10). The extension utility is firstly calculated for the ordered itemset I^* and the calculation procedure is given in Table 13. The resulting extension utilities for $I^* = \{d, b, a, e, c\}$ are {53, 36, 37, 27, 13}respectively, which are stored in the transaction array. Since only $eu(d) = 53 > 40 = minutil$, the rootwidthnode is $\{d\}$.

Step 5 (line 12). $\alpha = \{d\}$.

In lines 13 to 15, the utility of item α is calculated according to the RTD as shown in Table 14, that is $u(d) = 2 + 12 + 6 = 20 < 40 = minutil$. Therefore, item $\{d\}$ is not the HUI and nothing is output.

In line 16, according to Table 12, EOI(d) = {b, a, e, c} with the extension utility is calculated in Table 14. Since $eu(b) = 45 > 40 = minutil$, the widthnode is $\{b\}$.

In line 18, the calculation procedure of itemset TWU is the same as the extension utility.

For line 19, we call the search function to obtain the HUIs with the specified parameters. In this example, the parameters are ($\{d\}$, $\{b\}$, $\{b, c, e\}$, $\{(T_1, 2)(T_3, 6)(T_4, 12)\}$, 40).

For line 1 in Algorithm 2. (refer to A2.)). $i = 0$.

For line 2 in A2.), the item is EOI(d)[0] = b.

For line 3 in A2.), $\alpha = \{d, b\}$.

For lines 4 to 6 in A2.), according to Definition 1., $u(\{d, b\}) = u\{d, T_4\} + u\{d, T_3\} + u\{b, T_4\} + u\{b, T_4\} = 12 + 6 + 4 + 8 = 30 < 40 = minutil$, then $\{d, b\}$ is not HUI. Table 15 shows the calculation procedure of itemset TWU considering $\{c, e, a, b\}$.

In line 7 in A2.), $EOI(b) = \{a, e, c\} \cap \{b, c, e\} = \{e, c\}$. Then $eu(e)$ and $eu(c)$ can be calculated with the same procedure as shown in Table 16 with the results of 31 and 22, respectively, and neither of them is HUI. The width node is empty, and then the following step, line 10, does not need to be executed. Table 17 shows the calculation procedure of itemset TWU considering $\{c, e\}$.

Table 11 The RTD after pruning in Step 2

Items	ITDs
d	$(T_1, 2)(T_3, 6)(T_4, 12)$
b	$(T_3, 8)(T_4, 4)(T_5, 4)$
a	$(T_1, 5)(T_2, 10)(T_4, 5)$
e	$(T_2, 6)(T_3, 3)(T_4, 3)(T_5, 3)$
c	$(T_1, 1)(T_2, 6)(T_3, 3)(T_4, 1)(T_5, 2)$

Table 12 The EOI after pruning in Step 2

Item	EOI
d	b, e, a, c
b	a, e, c
a	e, c
e	c
c	

4.8 Time complexity analysis

The time costs of the HUIM-SU algorithm come from three processes, i.e., reorganising the transaction database, repeatedly pruning based on TWU, and the depth-first search. Let m be the number of transactions, l_t be the length of transactions, n be the number of items, l_e be the length of EOI, d_c be the depth of the construction tree, and l_i be the length of ITD. The time complexity of reorganising the transaction database is $O(m \cdot {l_t}^2)$. Repeated pruning based on TWU is performed in linear time as $(O(n \cdot l_i \cdot l_t))$. The time of the depth-first search is $\mathrm{O}(d_c \cdot n \cdot (l_i + l_e \cdot l_i))$. Thus, the time complexity of the HUIM-SU is $O(n \cdot l_i \cdot l_t + m \cdot {l_t}^2 + d_c \cdot n \cdot (l_i + l_e \cdot l_i))$.

5 Experimental results

Four utility-list-based state-of-the-art HUIM algorithms, the HUI-Miner, the FHM, the HUP-Miner, and the ULB-Miner are compared with the proposed HUIM-SU algorithm. Four groups of experiments are conducted to evaluate the performance of the algorithms. The first group of experiments investigate the number of candidates to compare the efficiency of the pruning strategy. The second and third groups are used to compare the memory usage and execution time w.r.t the different *minutil*. The fourth group of experiments is designed to examine the robustness of time growth w.r.t the number of transactions. All the experiments were carried out on a PC with a 4th generation Core i3 dual-core processor and 8GB RAM running on the Windows 10 operation system.

Table 13 The calculation procedure of extension utility

TA	T_1	T_2	T_3	T_4	T_5	Extension utility (eu)
Add ITD(c)	1	6	3	1	2	13
Add ITD(e)	–	12	6	4	5	27
Add ITD(a)	6	22	–	9	–	37
Add ITD(b)	–	–	14	13	9	36
Add ITD(d)	8	–	20	25	–	53

Table 14 The calculation procedure of extension utility with the rootwidenode d

TA	T_1	T_3	T_4	Extension utility (eu)
Initial values (ITD(d))	2	6	12	–
Add ITD(c)	3	9	13	23
Add ITD(e)	–	12	16	28
Add ITD(a)	8	–	21	29
Add ITD(b)	–	20	25	45

All algorithms are implemented in a Java environment (version 1.9.0_191). The measurements of time and memory are calculated using the standard Java API. The datasets and the codes of the compared algorithms are taken from spmf (http://www.philippe-fournier-viger.com/spmf/). The specific characteristics of the four sparse datasets (BMS, Foodmart, Retail, and Kosarak) and the four dense datasets (Chess, Connect, Pumsb, and Accident) are shown in Table 18.

5.1 Performance comparison on the number of candidates

The number of candidates is the sum of the width nodes that have been visited in the depth-first search process, which is influenced by the pruning strategies. The execution time is proportional to the number of the candidates. The experimental results are shown in Table 19. HUIM-SU is generally approximately two to three orders of magnitude better than all the other algorithms. From Table 18, it is clear that the dataset with the longest average transaction length is Pumsb. In general, the longer average transaction length results in the longer solution length, which means the mining depth is much deeper and greatly influences the pruning efficiency greatly. For the HUIM-SU algorithm, the pruning strategy prunes from both the width and the depth, which can reduce the number of candidates effectively and has shown its excellent performance.

5.2 Performance comparison on execution time w.r.t *minutil*

The execution times on each dataset by all the five algorithms with different *minutil* values are evaluated in this group of experiments. The value of *minutil* is reduced while running the algorithms until one of the algorithms is too slow, memory overflow occurs or a clear winner is evident [34]. The experimental results are shown in Figure 5 and 6. On BMS, the HUIM-SU is two to three orders of magnitude faster than the other four algorithms. On Chess, Connect, Foodmart, and Pumsb, the HUIM-SU is the fastest, followed by the ULB-Minier, which is

Table 15 The calculation procedure of itemset TWU

TU	T_1	T_3	T_4	itemset TWU ($itwu$)
Add ITD(c)	8	20	28	56
Add ITD(e)	–	20	28	48
Add ITD(a)	8	–	28	36
Add ITD(b)	–	20	28	48

Table 16 The calculation procedure of extension utility

TA	T_3	T_4	T_5	Extension utility (eu)
Initial values (ITD(b))	8	4	4	–
Add ITD(c)	11	5	6	22
Add ITD(e)	14	8	9	31

Table 17 The calculation procedure of itemset TWU

TA	T_3	T_4	T_5	Extension utility (eu)
Add ITD(c)	20	28	11	59
Add ITD(e)	20	28	11	59

Table 18 Dataset characteristics

Dataset	#Transaction	#Distinct Items	Avg.trans.length	Type
BMS	59,601	497	4.8	Sparse
Chess	3,196	75	37.0	Dense
Connect	67,557	129	43.0	Dense
Foodmart	4,141	1,559	4.4	Sparse
Pumsb	49,046	2,113	74	Dense
Retail	88,162	16,470	5.3	Sparse
Accident	340,183	468	33.8	Dense
Kosarak	990,000	41270	8.1	Sparse

Table 19 Number of candidates on different datasets (best results are in bold)

Dataset	$minutil$	FHM	HUI-Miner	HUP-Miner	ULB-Miner	HUIM-SU
BMS	2,280,000	2,604,258	25,605,274	2,546,443	2,604,260	**266**
Chess	650,000	9,227	9,635	8,613	9,227	**899**
Connect	16,500,000	25,191	24,320	24,305	24,320	**2,021**
Foodmart	5,000	53,955	6,696,197	78,183	54,133	**32,158**
Pumsb	12,500,000	80,2818	1,224,366	804,421	802,819	**34,513**
Retail	40,000	905	68,061	12,507	905	**418**
Accident	20,000,000	926	1,044	1,023	926	**57**
Kosarak	1,500,000	16,776	255,633	178,941	16,776	**638**

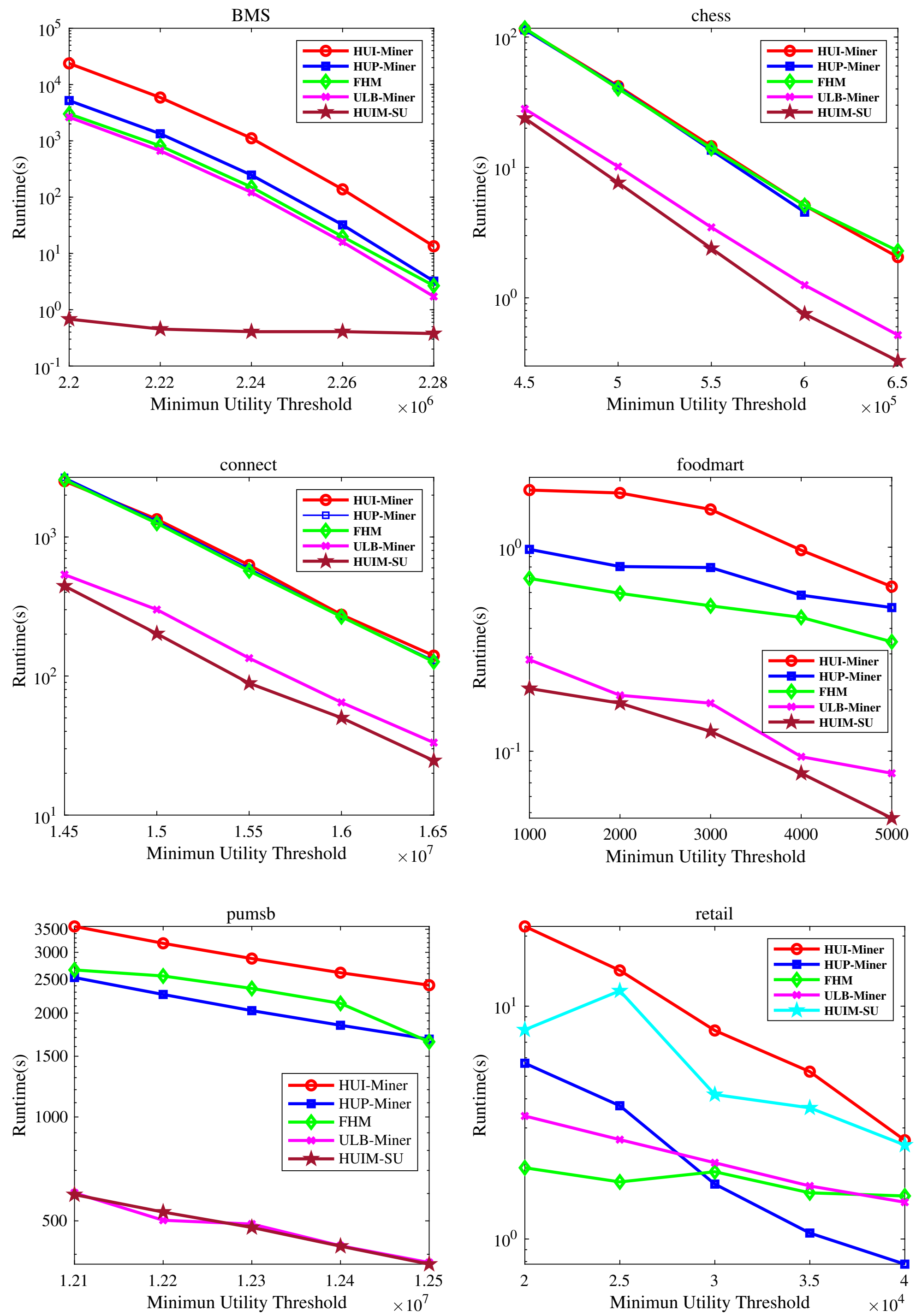
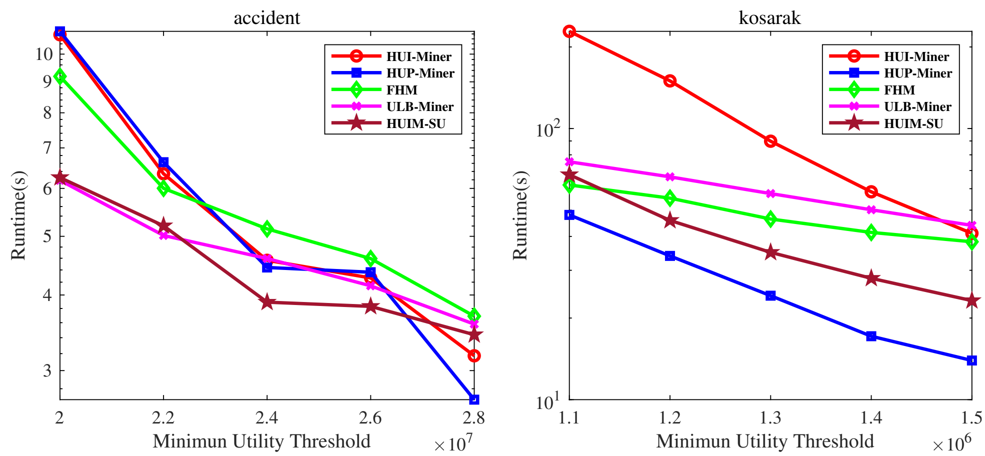


Fig. 5 Execution times on different datasets

approximately one order of magnitude greater than the other algorithms. On the accident and kosarak datasets, the HUIM-SU is slightly worse than the best algorithms. On retail, the HUIM-SU is much worse than the best one, which is occurs due to two reasons from the characteristics of the dataset and the proposed algorithm. From Table 18, it can be seen that the retail dataset is not large but has relatively more distinct items. For the proposed HUIM-SU, the execution time mainly costs the reorganising process, the pruning process, and the search procedure. When the number of distinct items is greater, the reorganising process and pruning process of the HUIM-SU cost much more time. If the dataset is not large, the total execution time is short for all the compared algorithms, while the reorganising process

Fig. 6 Execution times on different datasets (continue)



and the pruning process in the HUIM-SU account for a larger proportion of the execution time.

From the overall comparison results, the HUIM-SU has shown the best performance on the execution time for five out of eight datasets. There are two reasons for the effectiveness of the proposed HUIM-SU. First, the reorganised transaction database effectively reduces the calculation time of the utility of an item or an itemset. Second, the proposed pruning strategy based on extension utility and itemset TWU reduces the search space. The tighter upper bounds on the utility of the itemsets can reduce execution time by narrowing the search space.

5.3 Performance comparison on memory usage w.r.t *minutil*

Table 20 shows the maximum memory consumption of all five algorithms on eight datasets. With the BMS, retail, and kosarak datasets, the HUIM-Su substantially outperforms the other algorithms. On chess, foodmart, accident and pumsb, the HUIM-SU gets third place. On connect, HUIM-SU gets second place. The HUIM-SU ULB-Miner shows an overall better performance than the other four algorithms on memory usage. The maximum memory of the HUIM-SU in all eight datasets is 760 MB, which is the smallest in all algorithms. The HUIM-SU uses less than 100 MB memory on two datasets and less than 500 MB memory on three datasets. The simplified utility-list helps the HUIM-SU to reduce memory consumption, and the ITD, which is used to replace the utility-list structure, helps to use less memory. The transaction array in the HUIM-SU only creates a one-dimensional array during the pruning process, which also helps to reduce memory consumption.

5.4 Scalability

In this subsection, by varying the size of the transaction database from 25% to 100%, we study the scalability of the proposed algorithm. The experimental results are presented in Fig. 7. The $minutil$ values are 2200 K, 145500 K, 12100 K, and 20 K for the BMS, Connect, pumsb, and retail datasets, respectively. According to the figure, it is clear that as the size of the database is increased, the HUIM-SU achieves the best performance on the databases of BMS, connect,

Table 20 Maximum memory usage (MB) on all datasets (best results are in bold)

Dataset	HUIM-SU	FHM	HUI-Miner	ULB-Miner	HUP-Miner
BMS	**29**	668	668	423	91
Chess	419	674	530	**69**	102
Connect	760	1822	1831	**422**	1498
Foodmart	67	63	131	69	**58**
Pumsb	751	1670	1818	**458**	677
Retail	**121**	317	273	287	215
Accident	360	709	807	**283**	284
Kosarak	**588**	971	807	815	658

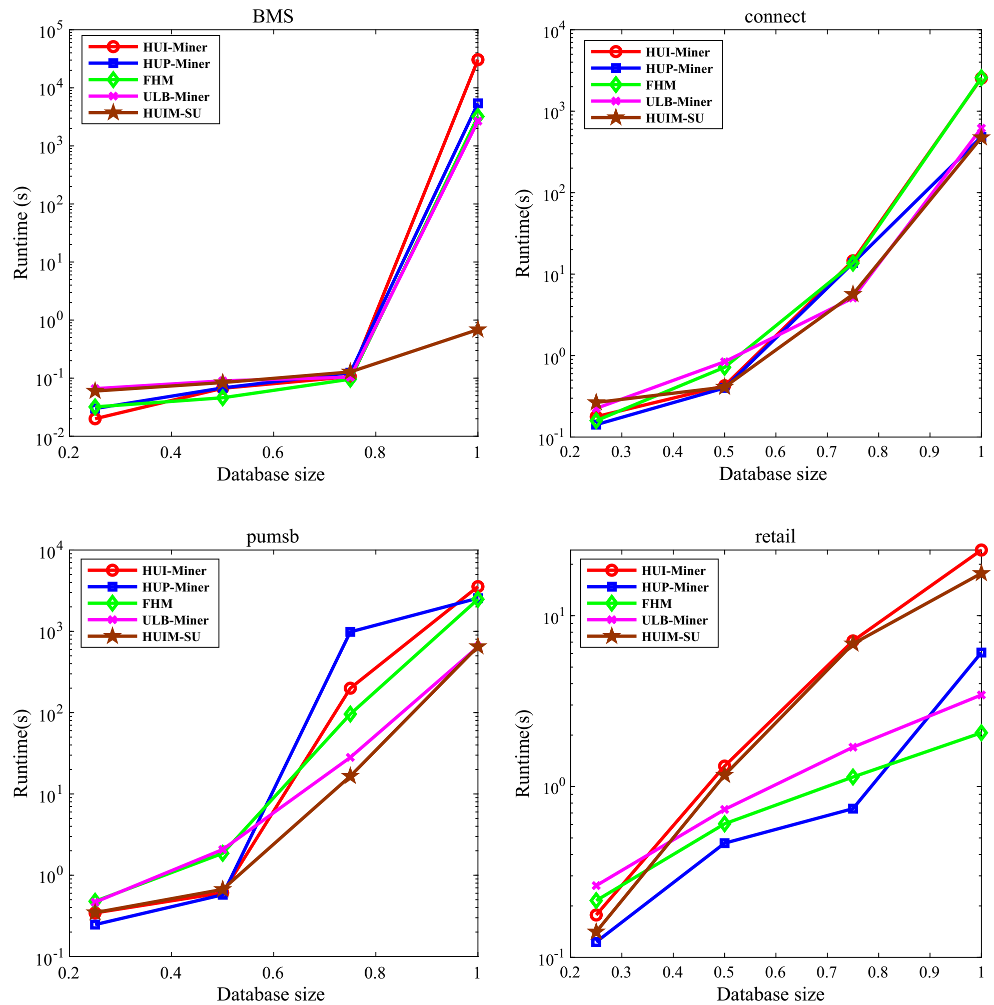


Fig. 7 Scalability of five algorithms

6 Conclusion

To address the problem that the utility-list-based HUIM algorithms may be time-consuming and memory consuming, an efficient HUIM algorithm based on the simplified utility-list is presented in this paper, which is named the HUIM-SU. With the simplified utility-list, a repeated pruning strategy based on TWU is employed in the HUIM-SU to effectively reduce the number of items. The construction tree is designed to represent the structure of items according to the TWUs and helps to reduce the search space. The compressed storage technique based on the definition of EOI is introduced to store the construction tree with less memory. Two upper bounds, which are the extension utility and the itemset TWU, are introduced to substantially prune the search space. The transaction array, which is a novel array-based approach, is designed in the HUIM-SU to calculate the two upper bounds easily and quickly. The experimental results obtained on dense and sparse datasets demonstrate that the proposed HUIM-SU shows better performance in terms of the number of candidates, memory usage, and execution time than the FHM, the HUP-Miner, the HUI-Miner, and the ULB-Miner. In the future, we will study more efficient search techniques for the HUIM-SU and test the performance on real-world bigdata scenarios.

Acknowledgements This work was supported in part by the National Key R&D Program of China under Grant 2017YFC1601800 and 2017YFC1601000, in part by the National Natural Science foundation of China under Grant 62073155, 62002137, 62106088, and 61673194, in part by "Blue Project" in Jiangsu Universities, China, in part by Guangdong Provincial Key Laboratory under Grant 2020B121201001, in part by Advanced Research Project of Specialty Leading Person in Higher Vocational Colleges in Jiangsu Province.

Author Contributions Zaihe Cheng: Methodology. Wei Fang: Supervision. Wei Shen: Software. Writing- Original draft preparation. Jerry Chun-Wei Lin, Bo Yuan: Resources, English language.

Funding This work was supported in part by the National Key R&D Program of China under Grant 2017YFC1601000 and 2017YFC1601800, in part by the National Natural Science foundation of China, under Grant 62073155, 62106088, 61673194, and 61672263.

Declarations

Ethics approval and consent to participate This work does not contain any studies with human participants performed by any of the authors.

Consent for Publication Informed consent was obtained from all individual participants included in this work.

Conflict of Interests The authors declare that they have no conflict of interest.

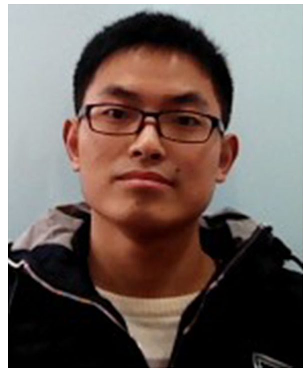
References

1. Luna JM, Fournier-Viger P, Ventura S (2019) Frequent itemset mining: a 25 years review. WIREs Data Mining and Knowledge Discovery 9(6):1329. https://doi.org/10.1002/wdm.1329. https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1329
2. Goyal P, Challa JS, Shrivastava S, Goyal N (2020) Anytime frequent itemset mining of transactional data streams. Big Data Research 21:100146. https://doi.org/10.1016/j.bdr.2020.100146
3. Xun Y, Cui X, Zhang J, Yin Q (2021) Incremental frequent itemsets mining based on frequent pattern tree and multi-scale. Expert Sys Appl 163:113805. https://doi.org/10.1016/j.eswa.2020.113805
4. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: International conference on very large data bases
5. Fournier-Viger P, Chun-Wei Lin J, Truong-Chi T, Nkambou R (2019) A Survey of High Utility Itemset Mining. In: Fournier-viger P, Lin JC-W, Nkambou R, Vo B, Tseng V. S. (eds) Springer, Cham, pp 1–45
6. Karagoz P, Cekinel RF (2019) High-utility pattern mining: theory, algorithms and applications. In: Studies in big data, 2019
7. Han X, Liu X, Li J, Gao H (2020) Efficient top-k high utility itemset mining on massive data. Inf Sci 557:382–406. https://doi.org/10.1016/j.ins.08.028
8. Gan W, Lin JC-W, Fournier-Viger P, Chao H.-C, Tseng VS, Yu PS (2021) A survey of utility-oriented pattern mining. IEEE Trans Knowl Data Eng 33(4):1306–1327. https://doi.org/10.1109/TKDE.2019.2942594
9. Amaranatha Reddy P, Hazarath Murali Krishna Prasad M (2021) High utility item-set mining from retail market data stream with various discount strategies using egui-tree. J Ambient Intell Human Comput, https://doi.org/10.1007/s12652-021-03341-3
10. Krishna GJ, Ravi V (2021) High utility itemset mining using binary differential evolution: An application to customer segmentation. Expert Sys Appl 181:115122. https://doi.org/10.1016/j.eswa.2021.115122
11. Kannimuthu S, Chakravarthy DG (2022) Discovery of interesting itemsets for web service composition using hybrid genetic algorithm. Neural Process Let, https://doi.org/10.1007/s11063-022-10793-x
12. Tseng VS, Shie BE, Wu CW, Yu PS (2013) Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772–1786
13. Zhang C, Du Z, Gan W, Yu PS (2021) Tkus: Mining top-k high utility sequential patterns. Inf Sci 570:342–359. https://doi.org/10.1016/j.ins.2021.04.035
14. Kim H, Yun U, Baek Y, Kim J, Vo B, Yoon E, Fujita H (2021) Efficient list based mining of high average utility patterns with maximum average pruning strategies. Inf Sci 543:85–105. https://doi.org/10.1016/j.ins.2020.07.043
15. Lin JC-W, Djenouri Y, Srivastava G, Yun U, Fournier-Viger P (2021) A predictive ga-based model for closed high-utility itemset mining. Appl Soft Comput 108:107422. https://doi.org/10.1016/j.asoc.2021.107422
16. Singh K, Singh SS, Kumar A, Shakya HK, Biswas B (2018) Chn: an efficient algorithm for mining closed high utility itemsets with negative utility. IEEE Trans Knowl Data Eng:1–1 (ealy access). https://doi.org/10.1109/TKDE.2018.2882421
17. Nam H, Yun U, Yoon E, Chun- Wei Lin J (2020) Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions. Inf Sci 529:1–27. https://doi.org/10.1016/j.ins.2020.03.030
18. Singh K, Singh SS, Kumar A, Biswas B (2019) Tkeh: an efficient algorithm for mining top-k high utility itemsets. Appl Intell 49(3):1078–1097. https://doi.org/10.1007/s10489-018-1316-x
19. Song W, Zheng C, Huang C, Liu L (2021) Heuristically mining the top-k high-utility itemsets with cross-entropy optimization. Appl Intell, https://doi.org/10.1007/s10489-021-02576-z
20. Dam TL, Li K, Fournier-Viger P, Duong QH (2017) An efficient algorithm for mining top- k on-shelf high utility itemsets. Knowl Inf Syst 52(3):1–35
21. Dawar S, Sharma V, Goyal V (2017) Mining top-k high-utility itemsets from a data stream under sliding window model. Appl Intell 47(4):1240–1255. https://doi.org/10.1007/s10489-017-0939-7
22. Fournier-Viger P, Zhang Y, Chun-Wei Lin J, Fujita H, Koh YS (2019) Mining local and peak high utility itemsets. Inf Sci 481:344–367
23. Truong T, Duong H, Le B, Fournier-Viger P (2020) Ehausm: an efficient algorithm for high average utility sequence mining. Inf Sci 515:302–323
24. Singh K, Kumar R, Biswas B (2022) High average-utility itemsets mining: a survey. Appl Intell 52(4):3901–3938. https://doi.org/10.1007/s10489-021-02611-z
25. Fournier-Viger P, Li Z, Lin JC-W, Kiran RU, Fujita H (2019) Efficient algorithms to identify periodic patterns in multiple sequences. Inf Sci 489:205–226
26. Ashraf M, Abdelkader T, Rady S, Gharib TF (2022) Tkn: an efficient approach for discovering top-k high utility itemsets with positive or negative profits. Inf Sci 587:654–678. https://doi.org/10.1016/j.ins.2021.12.024
27. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM international conference on information and knowledge management, CIKM '12, pp 55–64. Association for computing machinery, New York, https://doi.org/10.1145/2396761.2396773
28. Krishnamoorthy S (2015) Pruning strategies for mining high utility itemsets. Expert Syst Appl 42(5):2371–2381
29. Fournier-Viger P, Wu C-W, Zida S, Tseng VS (2014) Fhm: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: International symposium on methodologies for intelligent systems, pp 83–92
30. Duong Q-H, Fournier-Viger P, Ramampiaro H, Nørvåg K, Dam T-L (2018) Efficient high utility itemset mining using buffered utility-lists. Appl Intell 48(7):1859–1877. https://doi.org/10.1007/s10489-017-1057-2

31. Srikumar K (2017) Hminer: efficiently mining high utility itemsets. Expert Syst Appl 90:168–183
32. Aryabarzan N, Minaei-Bidgoli B, Teshnehlab M (2018) negfin: an efficient algorithm for fast mining frequent itemsets. Expert Syst Appl 105:129–143
33. Liu Y, Liao W, Alok C (2005) A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-asia conference on advances in knowledge discovery & data mining
34. Zida S, Fournier-Viger P, Lin JC-W, Wu C-W, Tseng VS (2015) Efim: a highly efficient algorithm for high-utility itemset mining. Adv Artif Intell Soft Comput 9413:530–546
35. Lin JC-W, Gan W, Fournier-Viger P, Hong T-P, Zhan J (2016) Efficient mining of high-utility itemsets using multiple minimum utility thresholds. Knowl-Based Syst 113:100–115. https://doi.org/10.1016/j.knosys.2016.09.013
36. Peng A, Koh YS, Riddle P (2017) mhuiminer: a fast high utility itemset mining algorithm for sparse datasets:196–207, https://doi.org/10.1007/978-3-319-57529-2_16
37. Vuong N, Le B, Truong T, Nguyen D-P (2021) Efficient algorithms for discovering high-utility patterns with strong frequency affinities. Expert Syst Appl 169:114464. https://doi.org/10.1016/j.eswa.2020.114464
38. Dawar S, Goyal V, Bera D (2017) A hybrid framework for mining high-utility itemsets in a sparse transaction database. Appl Intell 47(3):809–827
39. Hong G, Hong T-P, Tseng VS (2014) An efficient projection-based indexing approach for mining high utility itemsets. Knowl Infn Syst 38(1):85–107

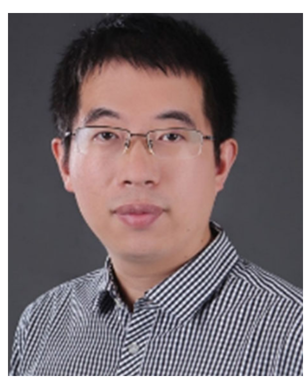
Zaihe Cheng received his Master degree in Jiangnan University in 2012. He is now a senior visiting scholar of Jiangnan University and an associate professor at School of Internet of Things, Wuxi Institute of Technology. His research interest is data mining and big data analysis.

Wei Fang received his PhD degree in Jiangnan University in 2008. He is now a professor of Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Jiangnan University. His research interest is evolutionary algorithm and its applications.



Wei Shen received his Master degree in Jiangnan University in 2020. His research interest is evolutionary algorithm and data mining.

Jerry Chun-Wei Lin received his Ph.D. from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan in 2010. He is currently a full Professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 360+ research articles in refereed journals (IEEE TKDE, IEEE TCYB, IEEE TII, IEEE TITS, IEEE TIAS, IEEE TETCI, IEEE SysJ, IEEE SensJ, IEEE IOTJ, ACM TKDD, ACM TDS, ACM TMIS, ACM TOIT, ACM TIST) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, PAKDD), 11 edited books, as well as 33 patents (held and filed, 3 US patents). His research interests include data mining, soft computing, artificial intelligence and machine learning, and privacy preserving and security technologies. He is the Editor-in-Chief of the International Journal of Data Science and Pattern Recognition, the Guest Editor/Associate Editor for several IEEE/ACM journals such as IEEE TFS, IEEE TII, ACM TMIS, ACM TOIT, and IEEE Access. He has recognized as the most cited Chinese Researcher respectively in 2018 and 2019 by Scopus/Elsevier. He is the Fellow of IET (FIET), senior member for both IEEE and ACM.



Bo Yuan received the B.Sc. degree and the Ph.D. degree in electronic information science and technology from University of Science and Technology of China (USTC), Hefei, China, in 2009 and 2014 respectively. During 2012/05 to 2013/04, he was a visiting student at School of Computer Science, University of Birmingham, Birmingham, UK. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China. His current research interests include evolutionary computation, electronic design automation, and machine learning.