Expert Systems With Applications 237 (2024) 121489

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

An efficient method for mining High-Utility itemsets from unstable negative profit databases

N.T. Tung [a], Trinh D.D. Nguyen [b], Loan T.T. Nguyen [c,d,*], Bay Vo [a]

[a] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Vietnam
[b] Faculty of Information Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, Vietnam
[c] School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam
[d] Vietnam National University, Ho Chi Minh City, Vietnam

ARTICLE INFO

Keywords:
Data mining
High-utility itemset mining
Negative profit
Unstable negative profits

ABSTRACT

The study of High-Utility Itemset Mining (HUIM) and Frequent Itemset Mining (FIM) is crucial since it explains consumer behavior and offers actionable advice to improve business results. HUIM algorithms have been successfully established to identify high-utility itemsets, including those with negative utilities. The problem with these approaches is that they presume incorrectly that items with negative utility across transactions would always be losses. Products with positive profitability may seem negative when combined with other items to increase sales or reduce inventory. Using strict upper-bound approaches, this paper presents strategies for making database scanning more efficient and reducing the number of prospective candidates. We also prove that it is correct to use the proposed upper-bounds for pruning on several types of items in the database. Based on all the proposed solutions, we develop a novel algorithm to solve this problem efficiently. To demonstrate their efficiency, the algorithms are tested against states-of-art HUIM algorithm on diverse datasets with regard to size and characteristics with unstable negative profits.

1. Introduction

Finding all the tuples that appear together in the database is the goal of Frequent Itemset Mining (FIM). This is a well-known problem in analyzing customer behavior from retail transaction databases. Itemsets are collections of tuples that occur together in a transaction, and their support is the number of times that they occur in the database. A Frequent Itemset (FI) is a collection of items where the frequency of recurrence does not fall below a defined minimum (user-specified threshold). FIM algorithms heavily rely on the downward closure property because of their efficiency. The support of an itemset is larger than or equal to that of its supersets, according to this property. This property is entirely dependent on an itemset's anti-monotonic property. FIM algorithms may be sped up with the use of the downward closure property. If an itemset is infrequent, the itemset and all its supersets may be eliminated from the search process without causing any harm. As a result, less time and space are spent computing and storing candidate results (Gan et al., 2019; Luna et al., 2019; Tian et al., 2022).

FIM is employed in a variety of practical applications and serves as the foundation for various applications of itemset mining, such as recommendation systems (Noaman et al., 2016). Association rules may be generated from FIM and are often utilized in providing useful suggestions. On the other hand, FIM has a significant disadvantage: it only considers the number of instances of an itemset in the database, which is the number of transactions containing that itemset. In fact, all items in a transaction database have an associated profit. Some products might be rarely purchased, yet their profit is immense, and FIM disregards this observation. High-Utility Itemset Mining (HUIM) was thus developed to locate all itemsets with a high profit (or utility). While FIM just detects frequent itemsets, HUIM can detect itemsets with high profit or revenue. The HUIM is thus a more extensive version of FIM.

Unlike FIM algorithms, which use a *minsup* threshold to investigate all itemsets containing at least enough occurrences of an item, HUIM methods use a *minU* threshold. This value is determined by the users and serves as an input to the HUIM algorithms. High-utility itemsets are defined as all itemsets in the database with a utility higher than or equal to *minU*. Numerous occurrences of an item may occur in a single transaction, with each occurrence representing a quantity (such as the

* Corresponding author.
E-mail addresses: nt.tung@hutech.edu.vn (N.T. Tung), 20126291.trinh@student.iuh.edu.vn (T.D.D. Nguyen), nttloan@hcmiu.edu.vn (L.T.T. Nguyen), vd.bay@hutech.edu.vn (B. Vo).

https://doi.org/10.1016/j.eswa.2023.121489
Received 16 April 2023; Received in revised form 17 August 2023; Accepted 5 September 2023
Available online 9 September 2023
0957-4174/© 2023 Elsevier Ltd. All rights reserved.

Table 1
An illustration of a transaction database.

TID	Items	Quantities
T_1	a, b, d, e, f, g	2, 2, 1, 3, 2, 1
T_2	b, c	1, 5
T_3	b, c, d, e, f	2, 1, 3, 2, 1
T_4	c, d, e	2, 1, 3
T_5	a, f	2, 3
T_6	a, b, c, d, e, f, g	2, 1, 4, 2, 1, 3, 1
T_7	b, c, e	3, 2, 2

number of units bought in a single invoice), and each item is linked to another integer indicating its unit value. (e.g., revenue, profit, etc.) (M. Liu & Qu, 2012). The utility of an item may be calculated by multiplying its cost and benefit (e.g., the profit gained from a customer purchase). The complexity of HUIM is significantly higher than FIM since the utility does not satisfy the downward closure property. It's possible that the supersets of a given set have a greater, equal, or lower utility than the set itself. In order to reduce the search space of an itemset's superset in HUIM, a new anti-monotonic upper-bound must be applied. (M. Liu & Qu, 2012).

Numerous HUIM algorithms have been created since the idea was initially proposed to enhance mining performance, such as DTWU-Mining (Le et al., 2011), HUI-Miner (J. Liu et al., 2012), FHM (Fournier-Viger et al., 2014), EFIM (Zida et al., 2017), H-Miner (Krishnamoorthy, 2017b), and MEFIM/iMEFIM (Nguyen et al., 2019). Besides, there are also algorithms to solve problems extended from HUIM, such as top-*k* HUIs (Han et al., 2021; Krishnamoorthy, 2019; Tseng et al., 2016), Cross-Level HUIs (Tung et al. 2022), Multiple-Level HUIs (Tung et al. 2022), HUIs from incremental data (Lee et al., 2018; Yun et al., 2019), HUIM and association rules (Mai et al., 2020; Sahoo et al., 2015). However, these algorithms all assume that the utility of items is a positive value, which is not practical in several real-world applications. In the real world, items can have negative utility. However, many HUIM algorithms cannot handle negative utility values. The fact that an item's utility is negative happens a lot in retail databases. For example, when products reaching their expiration date need a promotion to increase sales, stores often accept selling those items at a loss to recover capital or attract customers to buy other products at the store. This issue has been addressed by various algorithms, such as the HUINIV-Mine algorithm (Chu et al., 2009), FHN algorithm (Lin et al., 2016), EHIN algorithm (Singh et al., 2018) and EHMIN algorithm (Kim et al., 2022). These aim to extract itemsets from databases with negative utility. For illustration purposes, an example database with negative utility can be seen in Table 1 and Table 2.

Table 1 displays seven distinct transactions, each of which consists of two parts: the products involved and the quantities of those items. There is a direct correlation between each item in the database and its associated value, which is detailed in Table 2. It can be seen that the items b, f, and g contain negative utility values of -3, -1, and -2, respectively. Table 2 shows the profit for each item in the database introduced in Table 1. However, such a database negatively impacts the standard HUIM algorithms' upper bounds, as the negative utility value for item b decreases the upper bound, causing potential candidates to be overlooked.

Table 2
The table lists the profits for each item in the database.

Item	a	b	c	d	e	f	g
Profit	2	-3	1	4	1	-1	-2

Table 3
Example database with unstable negative profit.

TID	Items	Quantities	Profit
T_1	$a, \mathbf{b}, d, e, f, g$	2, 2, 1, 3, 2, 1	$-2, 1, 4, 1, -1, -2$
T_2	$\mathbf{b}, c$	1, 5	$-1, 1$
T_3	$\mathbf{b}, c, d, e, f$	2, 1, 3, 2, 1	$-1, 1, 4, 1, -1$
T_4	c, d, e	2, 1, 3	$1, 4, 1$
T_5	a, f	2, 3	$2, -1$
T_6	$a, \mathbf{b}, c, d, e, f, g$	2, 1, 4, 2, 1, 3, 1	$1, 1, 1, 4, 1, -1, -2$
T_7	b, c, e	3, 2, 2	$1, 2, 2$

1.1. Motivation

Traditional negative utility databases like those illustrated in Table 1 and Table 2 are employed by HUIM algorithms. The algorithms rely on the notion that the utility value of each item in each transaction is consistent across the board. However, in practice this is not the case since it is not sustainable for businesses to sell something that continually results in a loss. The return on items might be positive or negative depending on the circumstances of the sale, such as whether an item is sold individually or not. In a bundle, the item will still have a positive profit. Still, the item will have a negative profit when it needs to be sold at a discount in a combo offer to increase sales. However, there are still cases where items have negative utility throughout the database, depending on the store's orientation. To overcome these problems, the paper proposes a new type of database, as shown in Table 3.

The difference between the database in Table 3 and the database in Table 1 is that the utility of item b (in bold texts) is not stable at -3 but varied. It can be 1 in T_1, T_6 and T_7, and in T_2 and T_3, it is -1.

The database in Table 3 differs from that in Table 1 in that the value of item b (in bold) is not constant at -3 but varies. It may take the value 1 in T_1, T_6, and T_7, and the value -1 in T_2 and T_3.

HUIM algorithms with traditional (or fixed) negative profit databases are not designed to handle databases with unstable negative profits. This increases the challenge of the mining task. The key idea is that the negative-profit-capable HUIM algorithms split the items into two parts, one with negative utilities and the other with positive utilities. Items with negative utility will be placed further down the sorting order than those with positive utility. The utility of negative and positive items is determined independently. Algorithms like FHN and EHIN sort items with positive utility by their TWU values, and those with negative utility by their TWU values (or sorted in ascending order of support, including with GHUM and EHMIN). The more positive utility items appear, the more utility they have in the database, but the more negative utility items appear, the more they lose their utility. In addition, items with negative utility cannot stand alone, they must be combined with other items to create itemsets with positive utility. The complexity of the HUIM problem is increased with items of undetermined utility, and these items need to be processed separately. Therefore, HUIM algorithms designed to work with negative utility databases such as EHIN, FHN, GHUM and EHMIN cannot handle items containing both positive and negative profits.

1.2. Contributions

The main contributions of this paper include:

- This study presents a new database format and focuses on processing items with both positive and negative profits.
- The ordering of data items is also proposed. The research applies sophisticated strategies such as *redefined sub-tree* and *redefined local utility* to reduce the search space and achieve tighter upper bounds. The suggested upper bounds have been rigorously demonstrated to be accurate for trimming items with positive and negative profits, allowing them to safely and reliably exclude candidates from the search space.
- An algorithm called EMHUN (An **E**fficient method for **M**ining **H**igh-utility itemsets from **U**nstable **N**egative profit databases) is then

provided, and it concentrates on solving the HUIM problem on databases with unstable negative utilities, encompassing all the suggested features and techniques.

- The experiment results are executed in various databases to test the performance against the state-of-art algorithm. The tests included comparing execution time, memory consumption, the number of candidates, and the effect of the number of both positive and negative profits items on the performance of the two algorithms.

The rest of the paper is outlined as follows: The second part critically evaluates the relevant literature. The third part defines key terms and explains the problem being studied. In the fourth part, we explain the EMHUN method for extracting high-utility itemsets from an unstable profits database is described. In the fifth part, we evaluate the performance of the EMHUN and EHMIN algorithms on databases with unstable negative profits. This evaluation includes assessing memory usage, candidate generation, and analyzing the variation of execution time as the number of items with both negative and positive profits changes. (EHMIN is the newest algorithm in this mining category, and we modified it to work with the proposed database). Further research directions are discussed in the paper's conclusion.

2. Related work

2.1. High-utility itemset mining

Since Yao and Hamilton introduced the HUIM task, several advances have been made to increase its mining performance. Liu et al. devised the Two-Phase method (Y. Liu et al., 2005). In this, the Transaction Weight Utility (TWU) is a new upper bound that meets the downward closure property, allowing the elimination of unproductive candidates from the search space, resulting in the complete extraction of HUIs. However, the Two-Phase approach requires numerous scans of the database to identify all the HUIs, and the TWU is regarded as a secure but not tight upper bound, resulting in a longer processing time. Le et al. introduced the DTWU-Mining (Le et al., 2011) techniques using the diffset approach to reduce the number of candidates and database scans while keeping search time and memory use to a minimum. The UP-Growth algorithm (Tseng et al., 2010) was proposed in 2010 and is based on the FP-Growth algorithm. It uses four fundamental strategies to reduce the number of unpromising candidates that are checked: Discarding Global Unpromising Items and Node Utilities, Discarding Local Unpromising Items, and Decreasing Local Node Utilities. Qu et al. advanced the tree mining technique in 2023 by proposing the Hamm algorithm (Qu et al., 2023) to enhance the HUIM process in the tree and the novel TV (prefix Tree and utility Vector) structure that mines high-utility itemsets in a single phase without the need for candidate generation.

Using the Utility-List data format and a more stringent upper bound than TWU, dubbed the remaining utility, Liu et al. introduced the HUI-Miner method (M. Liu & Qu, 2012). The step of producing several candidates is no longer necessary since the Utility-List structure allows the algorithm to search the database just once and then combine the Utility-List of itemsets to generate bigger itemsets. In 2014, Fournier-Viger et al. proposed the FHM algorithm (Fournier-Viger et al., 2014) to enhance the performance of the HUI-Miner algorithm by reducing itemsets generated by two items through a strategy called Estimated Utility Co-occurrence Pruning (EUCP) and a data structure called EUCS (Estimated Utility Co-occurrence Structure). In 2017, Krishnamoorthy introduced the HUP-Miner (Krishnamoorthy, 2015) technique to improve upon the FHM algorithm by suggesting more efficient pruning methods to reduce the search space even more. The ULB-Miner method, released (Q.-H. H. Duong et al., 2018), makes use of a utility-list buffer structure to reduce both execution time and memory consumption. In 2017, Krishnamoorthy introduced the HMiner algorithm, which incorporates a number of pruning methods (TWU-prune, EUCS-prune, U-prune, LA-prune, and C-prune) and a proposed mechanism for merging duplicate transactions. The suggested method reduces the amount of time it takes to run the application by keeping "closed" transactions in a centralized location. The UBP-Miner (Wu et al., 2022) algorithm, developed by Wu et al. in 2022, is meant to improve upon the List-based approach by reducing the number of required transaction scans. The UBP-Miner algorithm has shown superior performance over the HUI-Miner and ULB-Miner algorithms in benchmark tests.

In addition to the utility-list and tree-based methods, the algorithms deal with partially identical transactions. One of the first algorithms based on this idea was EFIM. The algorithm was proposed (Zida et al., 2017). The algorithm uses a technique to merge similar transactions and rearrange items in transactions so they can be easily projected and merged to reduce memory usage. Besides, the algorithm also makes use of two rigorous and tight upper bounds: the sub-tree and the local utility. However, EFIM has a problem with high database scanning costs. To overcome this, Nguyen et al. proposed the iMEFIM algorithm (Nguyen et al., 2019), the authors proposed the P-Set structure to store the location of the necessary transactions. The algorithm also has a method to handle databases with dynamic utility per transaction.

In addition to the traditional HUIM algorithms, there are also other problems based on the HUIM problem that are common in the real world, such as Utility-Oriented Pattern Mining (Gan et al., 2021), Closed-HUIs (Dam et al., 2019; Tseng et al., 2015; Vo et al., 2020), Maximal-HUIs (H. Duong et al., 2022; Nguyen et al., 2020), HUIs in the incremental databases (Lee et al., 2018; Yun et al., 2019), uncertainty HUIs (Ahmed et al., 2021), top-k HUIs (Han et al., 2021; Krishnamoorthy, 2019; Tseng et al., 2016), Cross-Level HUIs (Tung et al., 2022a), Multiple-Level HUIs (Tung et al., 2022b), High-Utility Occupancy Pattern Mining (Gan et al., 2020) or matters of great interest such as Fuzzy Utility (FU) pattern mining (Ryu et al., 2022) and weighted high-utility pattern mining (Srivastava et al., 2021).

2.2. Mining high-utility itemsets with negative profit

In 2009, the first suggestion for a HUIM algorithm to run on a database containing losses was made. The algorithm named HUINIV-Mine (Chu et al., 2009) is based on Two-Phase approach. Although it is possible to perform the HUIM process on a database with negative utility, the HUINIV-Mine algorithm consumes a lot of time due to the generation of too many candidates. Then, Li et al. released FHN (Lin et al., 2016), an expansion algorithm of FHM that made use of the same technologies as FHM, including EUCS and EUCP. LA-Prune is more productive than HUINIV-Mine and uses less RAM while mining. To further leverage HUIs on datasets with negative profitability, Subramanian et al. introduced the UP-GNIV (Subramanian & Kandhasamy, 2015) approach based on the FP-Growth algorithm. Xu et al. developed the HUSP-NIV (Xu et al., 2017) approach so that high-utility sequence itemsets may be used even if they include negative profit items. The GHUM algorithm (Krishnamoorthy, 2017a) uses a utility-list structure as opposed to FHN's approach of dividing positive and negative items into distinct lists. It is evident that the GHUM algorithm is superior to the FHN algorithm after comparing the two. To mine HUIs containing negative items, Singh et al. published the EHIN algorithm (Singh et al., 2018). EHIN can potentially outperform FHN because it uses metrics developed for EFIM, such as local-utility and sub-tree utility, as well as strategies like projection and combining related transactions. The EHNL approach (Singh et al., 2019) was suggested to mine HUIs with negative items and restrict the size of an itemset. Sun et al. developed the THN algorithm (Sun et al., 2021), which employs the same metrics as EHIN and uses a strategy of raising the threshold with each itemset discovered. The Top-k HUIs containing negative items may also be mined using the TKN method, which was suggested (Ashraf et al., 2022). The suggested technique is adaptable to both positive and negative items thanks to the metrics and strategies utilized in the algorithm. Kim et al. introduced the EHMIN algorithm (Kim et al., 2022), and this is a list-based method that outperforms GHUM and FHN in experimental settings. The TKHUIM-GA algorithm (Luna et al., 2023) was introduced to mine the top-k HUIs, and it employs a genetic approach to decrease mining time and memory usage. In addition, the TKHUIM-GA method may be used to evaluate goods with a negative utility.

While several methods have been proposed to address issues with unfavorable elements in HUIM, the negative utility of items is assumed to be constant throughout the database by all such approaches, and this is not in line with how things work in the real world.

3. Preliminaries

To help solve the issue, we present several terms in this section.

Definition 1. Transaction database (Singh et al., 2018).

A transaction database, denoted as $D = \{T_1, T_2, \ldots, T_n\}$, is a collection of transactions. Each transaction in D, denoted as T_q, is a collection of k items from the set $I = \{i_1, i_2, i_3, \ldots, i_m\}$, which is a list of all m distinct items in the database. The unit profit value of the profit of each item i_j $(1 \le j \le m)$ in the transaction $T_k (1 \le k \le n)$ is denoted by the symbol $p(i_j, T_k)$. Moreover, $q(i_j, T_k)$ represents the amount of each item i_j in transaction T_k. The value of the utility of item i_j in transaction T_k is denoted by $u(i_j, T_k)$, and it is calculated as:

$$u(i_j, T_k) = p(i_j, T_k) \times q(i_j, T_k)$$

For example, the database shown in Table 3 contains 7 transactions. There are also 7 distinct items a, b, c, d, e, f and g. Transaction T_1 contains 6 items including a, b, d, e, f, g. In which, item b has the quantity of 2 and unit profit of each unit b is 1. Hence, the utility of b in T_1 is calculated by the formula:$u(b, T_1) = p(b, T_1) \times q(b, T_1) = 1 \times 2 = 2$.

Definition 2. Utility of an itemset in a transaction (Singh et al., 2018).

The utility of an itemset in a transaction T_q is the total of the utilities of its constituent items. The formula for determining the worth of an itemset in a given transaction is as follows, with the itemset represented by the notation $u(X, T_q)$

$$u(X, T_q) = \sum_{i_j \in X \wedge i_j \in T_q} (i_j, T_q)$$

For example: $u(bd, T_1) = u(b, T_1) + u(d, T_1) = (2 \times 1) + (1 \times 4) = 6$.

Definition 3. Utility of an itemset in the database (Singh et al., 2018).

In database D, the utility of an itemset is calculated as the total of its utility in all transactions containing the itemset. The following formula may be used to determine the utility of the itemset X of items in database D, denoted by the symbol $u(X)$.

$$u(X) = \sum_{X \subseteq T_q} u(X, T_q)$$

For example: $u(ab) = u(ab, T_1) + u(ab, T_6) = -1 + 3 = 2$;

$$u(bd) = u(bd, T_1) + u(bd, T_3) + u(bd, T_6) = 6 + 10 + 9 = 25$$

Definition 4. The problem of high-utility itemset mining (Singh et al., 2018).

Given an existing *minU* threshold set by the user and a transaction database (D), the goal of high utility itemsets mining is to find all itemsets that have the utility greater than or equal to a certain threshold, called *minU*.

4. EMHUN algorithm presentation

Definition 5. Transaction utility and redefined transaction utility (Singh et al., 2018).

Given a transaction T_k, the transaction utility of transaction T_k is calculated as the sum of the utility of the items in that transaction. It is denoted by $TU(T_k)$ and is calculated by the formula:

$$TU(T_k) = \sum_{i_j \in T_k} u(i_j)$$

It is not reasonable to use a transaction utility value in the HUIM problem with a database containing negative utility values as they reduce the utility of the whole transaction and can miss candidates. Therefore, to calculate transaction utility property the Redefined Transaction Utility was suggested, denoted as $RTU(T_k)$ and determined by the formula:

$$RTU(T_k) = \sum_{i_j \in T_k \wedge u(i_j) \rangle 0} u(i_j)$$

For example, considering transaction T_2 in Table 3:

$TU(T_2) = 1 \times (-1) + 5 \times 1 = 4;$

$$RTU(T_2) = 5 \times 1 = 5$$

Definition 6. Transaction Weight Utility and Redefine Transaction Weight Utility (Singh et al., 2018).

The Transaction Weight Utility of an itemset in a database D is the sum of the utility of all transactions that include that itemset. $TWU(X)$ is the Transaction Weight Utility, and it is defined as follows:

$$TWU(X) = \sum_{X \subseteq T_k \wedge T_k \in D} TU(T_k)$$

In the HUIM issue, TWU is used in the same way that TU is used when the transaction has negative utility. However, the Redefine Transaction Weight Utility is implemented to prevent candidates from being lost. The formula for determining $RTWU(X)$ is:

$$RTWU(X) = \sum_{X \subseteq T_k \wedge T_k \in D} RTU(T_k)$$

This is a crucial definition that is utilized by various HUIM algorithms to exclude unsuitable candidates from the search.

Property 1. Prune itemsets using the Redefine Transaction Weight Utility value.

If $RTWU(X)$ is less than the $minU$, then itemset X and all its supersets are not HUI and will be eliminated from the search area.

Let itemset X, and itemset Y such that $X \subset Y$, $RTWU(Y) \le RTWU(X)$

Proof: If the database contains items that are only negative and positive on the whole transaction, the property is proven in (Singh et al., 2018).

If the database contains three types of items, including those containing positive utility on the whole database, having negative utility on the whole database, and carrying both positive and negative profits values.

1.1. Given an itemset X. The negative utility of X in transaction T_k is $nu(X, T_k)$ and the positive utility of X in transaction T_k is $pu(X, T_k)$. $u(X, T_k) = pu(X, T_k) + nu(X, T_k)$ then $pu(X, T_k) \ge u(X, T_k)$ because $nu(X, T_k) \rangle 0$. $u(X) = \sum_{X \subseteq T_k \wedge T_k \in D} pu(X, T_k) + nu(X, T_k)$ and $u(X) \le pu(X)$.

Since $RTWU(X) = \sum_{X \subseteq T_k \wedge T_k \in D} RTU(T_k) = \sum_{X \subseteq T_k} (\sum_{i_j \in (T_k / X) \wedge u(i_j) \rangle 0} u(i_j) + pu(X, T_k)) \ge u(X)$ if $RTWU(X) < minU$ then X not a HUI.

1.2. Let $G(X)$ be the list of all transactions containing X, and $G(Y)$ be the list of all transactions containing Y. Because of $X \subset Y$ and $G(Y) \subseteq G(X)$, $RTWU(X) \ge RTWU(Y) \ge pu(Y) \ge u(Y)$. If itemset X has $RTWU(X) < minU$ then X and all supersets formed from X will not be HUIs and can be safely pruned out of the search space.

Table 4
RTWU results for each item in the example database.

Items	*a*	*b*	*c*	*d*	*e*	*f*	*g*
RTWU	29	56	56	49	60	44	25

For example, in the transaction in Table 3 :

$$RTWU(ab) = RTU(ab, T_1) + RTU(ab, T_6) = 9 + 18 = 27$$

Assuming $minU = 40$, then all itemsets that are supersets of $\{ab\}$ are not HUIs and will be removed from the search space.

Definition 7. Order of items in the database.

In traditional negative-utility-based HUIM algorithms, there are two types of items in the database: items with either positive or negative utility in the whole database. Items with positive utility will be prioritized before items with negative utility. However, when considering items with unstable utility, there are 3 types of items: items with positive utility only, items with negative utility only, and items that can either have positive or negative utility (or referred to as hybrid items).

To deal with this problem, we propose the following.

- Throughout the database, items with only positive utility will always be prioritized to be sorted first, followed by hybrid items, and finally, items with negative utility.
- Items in the above groups have been ordered from lowest to highest *RTWU*.

These rules ensure that items with positive utility are always discovered before items with negative utility.

For example, the *RTWU* findings for each item are shown in Table 4 below, based on the data in Table 3.

There are three lists from the database, including:

- Positive-utility-only items: c, d, e.
- Hybrid item: a, b.
- Negative-utility-only items: f, g.

Order of items by definition: $d \succ c \succ e \succ a \succ b \succ g \succ f$.

Definition 8. Remaining utility and Redefined remaining utility.

Given a transaction T_k, an itemset X, and a sort order $\succ$. The remaining utility of an itemset in transaction T_k is the sum of the utility of the items that are after all the items of itemset X in the ordered T_k. The remaining utility of an itemset X in the transaction T_k is denoted by $ru(X, T_k)$ and is calculated by the formula :

$$ru(X, T_k) = \sum_{i_j \in E(X, T_k)} u(i_j, T_k)$$

with the set of all items after X denoted as.

$E(X, T_k) = \{i | i \in T_k \wedge i \succ w, \forall w \in X\}$(M. Liu & Qu, 2012).

. Similar to the definition of remaining utility, the definition of Redefined remaining utility includes the utility of all items with a utility not less than 0 after the ordered transaction. The Redefined remaining utility of an itemset X in the ordered transaction T_k is denoted by $rru(X, T_k)$ and is calculated by the formula:

$$rru(X, T_k) = \sum_{i_j \in RE(X, T_k)} u(i_j, T_k)$$

with $RE(X, T_k) = \{i | i \in T_k \wedge i \succ w, \forall w \in X \wedge u(i_j, T_k) \rangle 0\}$.

For example, based on the information provided in Table 3 and the alphabetical arrangement of the items within, then:

$$ru(bd, T_3) = 2 \times 1 + (-1) \times 1 = 1$$

$$rru(bd, T_3) = 2 \times 1 = 2$$

Definition 9. Redefined remaining utility upper-bound.

An itemset X on a given database D has a Redefined remaining utility equal to the sum of the Redefined remaining utilities of all transactions in which X appears. Denoted by the symbol $rru(X)$, the Redefined remaining utility of an itemset X on database D is found using the formula:

Table 5
Result of database projection in..$\{bd\}$

TID	Items	Profits
T_1	e,f,g	3, −2, −2
T_3	e,f	2, −1
T_6	e,f,g	1, −3, −2

$$rru(X) = \sum_{X \subseteq T_k} rru(X, T_k)$$

An itemset X on database D has a Redefined remaining utility upper-bound equal to the sum of its Redefined remaining utility and its utility on database D. The Redefined remaining utility upper-bound is denoted by $rreu(X)$ and is defined as

$$rreu(X) = \sum_{X \subseteq T_k} rru(X, T_k)$$

The following property is critical for pruning the search space based:

Property 2. Prune itemsets by Redefined remaining utility upper-bound value.

Given itemset X and itemset Y such that $X \subset Y$, then $rreu(X) \geq rreu(Y)$. If itemset X has $rreu(X) < minU$ then all supersets formed from X will not be HUIs and can be safely pruned out of the search space.

Proof: If the database contains items that are only negative or only positive on the whole database, the property is proven in (Singh et al., 2018).

Briefly, let Y be a superset of X and i be an item such that $Y/X = i$. If i is a negative or positive item on the whole database, then this property is proven in (Singh et al., 2018). Our work only proves this property based on the hybrid items existing in the database.

For all transactions T in the database such that $Y \subseteq T$.

$$X \subset Y \subseteq T \Rightarrow Y/X \subseteq T/X$$

$$u(Y, T) = u(X, T) + u(i, T)$$

If $u(i, T) \rangle 0$ then $i \in (T/X \wedge u(i_k, T) > 0)$, and in contrast $i \notin (T/X \wedge u(i_k, T) > 0)$. $u(i, T) \leq \sum_{i_k \in (T/X \wedge u(i_k, T) > 0)} u(i_k, T)$ in any case. $u(X, T) + u(i, T) \leq u(X, T) + \sum_{i_k \in (T/X \wedge u(i_k, T) > 0)} u(i_k, T) = u(X, T) + rru(X, T)$.

Let $G(X)$ be all transactions containing X, $G(Y)$ be all transactions containing Y. Since $X \subset Y$ then $G(Y) \subseteq G(X)$.

$$u(Y) = \sum_{T \in G(Y)} u(Y, T) \leq \sum_{T \in G(Y)} u(X, T) + rru(X, T) \leq \sum_{T \in G(X)} u(X, T) + rru(X, T)$$

For example, based on the information provided in Table 3 and the alphabetical arrangement of the items within, then:

$$\begin{aligned} rru(bd) &= rru(bd, T_1) + rru(bd, T_3) + rru(bd, T_6) \\ &= u(e, T_1) + u(e, T_3) + u(e, T_6) = 3 + 2 + 1 = 6 \end{aligned}$$

$$rreu(bd) = u(bd) + rru(bd) = 6 + 25 = 31$$

Assuming that $minU = 35$, then bd and all supersets of bd(as $bdc, bde, bdf, bdg, andsoon$) are not HUIs and are removed from the search space.

Definition 10. Transaction projection on an itemset.

Let there be an ordered transaction T_k, itemset X, the projection of a transaction T_k using itemset X is denoted and $(T_k)_X$ defined as:

Table 6
Result of database projection in $\{bd\}$ and merging.

TID	Items	Profits
T_1'	e,f,g	4, −5, −4
T_3	e,f	2, −1

Table 7
The database results in Table 3 after transactions are sorted.

TID	Items	Profits
T_6	a,b,c,d,e,f,g	2, 1, 4, 8, 1, −3, −2
T_1	a,b,d,e,f,g	−4, 2, 4, 3, −2, −2
T_3	b,c,d,e,f	−2, 1, 12, 2, −1
T_5	a,f	4, −3
T_4	c,d,e	2, 4, 3
T_7	b,c,e	3, 4, 4
T_2	b,c	−1, 5

$$(T_k)_X = \{i_j | i_j \in T_k \wedge i_j \succ w, \forall w \in T_k\}$$

For example, given the transaction T_3 in the database in Table 3, $(T_3)_{\{bd\}} = \{e,f\}$.

Definition 11. Database projection on an itemset.

The set of all transactions projected by itemset X in the database is called the database projected by X, denoted by D_X and computed as:

$$D_X = \{(T_k)_X | T_k \in D \wedge (T_k)_X \neq \varnothing\}$$

For example, given the database in Table 3, projected database $D_{\{bd\}}$ is shown in Table 5:

Definition 12. Transaction merging using the same projection in the database.

Transaction merging is the task of combining similar transactions or transactions with the same projection into a single transaction to reduce the amount of data to be scanned.

For example, consider the database in Table 3, database $D_{\{bd\}}$ after merging T_1 and T_6 to become $T_1^{'}$ and is presented in Table 6:

Definition 13. Transaction sorting (Singh et al., 2018).

The transactions in database D will be sorted in the order defined below to improve the efficiency of the transaction projection and merging. For each transaction, all the items it contains are already sorted in the order presented in Definition 7. Each transaction will be traversed backward to be sorted in the order as defined in (Singh et al., 2018).

To order transactions in the database, it is necessary to know the order of the comparison process of two transactions. Suppose to compare two transactions T_x has j elements, and T_y has k elements to follow the following rules:

- Start checking the j^{th} element in transaction T_x and the kth element in T_y.
- If these two elements are the same, the comparison will continue with the $(j-1)^{th}$ and $(k-1)^{th}$ elements and continue until a position is found where the two elements in question have different values.
- If position p is found there such that the $(j-p)$ position in T_x is different from the element at position $(k-p)$ in transaction T_y. The element in the transaction that comes first in the previously specified item sort order, that transaction will be ranked second.
- If in the process of finding the difference between two transactions, the transaction that is finished first, that transaction will be ranked later.

For example: based on the information provided in Table 3 and the alphabetical arrangement of the items within, comparing two transactions $T_1 = \{a,b,d,e,f,g\}$ and $T_6 = \{a,b,c,d,e,f,g\}$, the steps are as follows:

- Starting with the last two elements of each transaction, both of which are g, the subsequent element will be the one preceding g in each respective transaction. This pattern persists with both elements continuing to be f Successively, items f and d undergo consideration, ultimately concluding at the 5^{th} position from the end of the transaction.
- In the case of transaction comparison, where transaction T_1 includes item b and transaction T_6 includes item c, the following occurs. Utilizing the alphabetical order, with b preceding c, the transaction encompassing c (T_6) is positioned ahead of the transaction containing b (T_1). The outcomes of this sorting process are outlined in Table 7.

The objective of sorting transactions is to enhance the efficiency of the transaction merging process. More specifically, the sorting process facilitates the proximity of projected transactions during merging. Consequently, when projected with an itemset, transactions sharing identical outcomes are positioned closely to each other, allowing for efficient merging.

- For example, the database in Starting with the last two elements of each transaction, both of which are g, the subsequent element will be the one preceding g in each respective transaction. This pattern persists with both elements continuing to be f Successively, items f and d undergo consideration, ultimately concluding at the 5th position from the end of the transaction.
- In the case of transaction comparison, where transaction $\mathrm{T_1}$ includes item b and transaction $\mathrm{T_6}$ includes item c, the following occurs. Utilizing the alphabetical order, with b preceding c, the transaction encompassing c ($\mathrm{T_6}$) is positioned ahead of the transaction containing b ($\mathrm{T_1}$). The outcomes of this sorting process are outlined in Table 8.

Table 7 after being projected by d will get the results in Table 8. Two transactions T_1 and T_6 after being projected against d already containing the same items and can be merged to get results in Table 9.

Definition 14. Redefined local utility of an itemset.

The redefined local utility of an itemset X and an item z is denoted by $rlu(X,z)$ and is calculated by the formula:

$$rlu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + rru(X,T_k)]$$

To reduce the search space, we utilize the Redefined local utility as a ceiling based on the following property:

Property 3. Prune itemsets by Redefined local utility value.

Given an itemset X and an item z such that $z \in \{z | z \succ w, \forall w \in X\}$. The relationship $rlu(X,z) \geq u(X \cup \{z\})$ holds.

Proof: If z has positive utility over the entire database, then the property has been proven (Zida et al., 2017).

- $u(X \cup \{z\}) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} u(X \cup \{z\}, T_k) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X, T_k) + u(z, T_k)]$.

Table 8
The result of the database in Table 7 after being projected by..d

TID	Items	Profits
T_6	e,f,g	1, −3, −2
T_1	e,f,g	3, −2, −2
T_3	e,f	2, −1
T_4	e	3

Table 9
The result of the database is projected by d after merging T_6 and..T_1

TID	Items	Profits
$T_6{'}$	e,f,g	4, −5, −4
T_3	e,f	2, −1
T_4	e	3

Table 10
The results of $RSU(X,i)$ and $RLU(X,i)$ for each item i, with $X = \{bd\}$ and alphabetical order.

Items	e	f	g
RLU	31	31	19
RSU	31	19	11

- If z is an item that has negative utility, the redefined local utility of X with z is $rlu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq k} [u(X,T_k) + rru(X,T_k)]$. Because, in any transaction T_k where $\{X \cup z\} \subseteq T_k$, $u(z,T_k) \rangle 0$.

$$rru(X,T_k) = \sum_{i_j \in T_k \wedge i_j \in ((T_k/X) - z)) \wedge u(i_j) > 0} u(i_j) \geq 0 > u(z,T_k)$$

For example: based on the information provided in Table 3 and the alphabetical arrangement of the items within, then:

$$\begin{aligned} rlu(b,d) &= [u(b,T_1) + rru(b,T_1)] + [u(b,T_3) + rru(d,T_3)] + [u(b,T_6) \\ &\quad + rru(b,T_6)] \\ &= 2 + 7 + (-2) + 15 + 1 + 13 = 36 \end{aligned}$$

$$u(bd) = u(bd,T_1) + u(bd,T_3) + u(bd,T_6) = 6 + 10 + 9 = 25$$

$$rlu(bd,f) \rangle u(bdf)$$

Definition 15. Redefined sub-tree utility of an itemset.

The redefined sub-tree utility of an itemset X and an item z is denoted by $rsu(X,z)$ and is calculated by the formula:

$$rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + u(z,T_k) + rru(z,T_k)]$$

To reduce the search space, we utilize the Redefined sub-tree utility as a ceiling based on the following property.

Property 4. Prune itemsets by the Redefine sub-tree utility value.

Given an itemset X and an item z such that $z \in \{z | z \succ w, \forall w \in X\}$. The relationship $rsu(X,z) \geq u(X \cup \{z\})$ holds.

Proof: If z is an item that has positive utility over the entire database, then the property has been proven (Zida et al., 2017).

- $u(X \cup \{z\}) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} u(X \cup \{z\}, T_k) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X, T_k) + u(z,T_k)]$
- If z is an item that has negative utility, the redefined sub-tree utility of X with z is $rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + u(z,T_k) + rru(z,T_k)] = u(X \cup \{z\}) + \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} rru(z, T_k)$. And $rru(X,T_k) \geq 0$ in any case.$rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + u(z,T_k) + rru(z,T_k)]$. Because $rru(z,T_k) = 0$

$$rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + u(z,T_k)] = u(X \cup \{z\})$$

We can adopt the two proposed and proven properties to prune complete unpromising branches of the set-enumeration tree (Singh et al., 2018). All itemsets X and single items z with an upper bound $rsu(X,z) \langle minU$ or $rlu(X,z) \langle minU$ are not promising candidates and will be pruned from the search space.

For example: based on the information provided in Table 3 and the alphabetical arrangement of the items within, then:

$$\begin{aligned} rsu(b,d) &= [u(b,T_1) + u(d,T_1) + rru(d,T_1)] + [u(b,T_3) + u(d,T_3) \\ &\quad + rru(d,T_3)] + [u(b,T_6) + u(d,T_6) + rru(d,T_6)] = [2+4+3] \\ &\quad + [(-2) + 12 + 2] + [1 + 8 + 1] = 31 > u(bd) = 25 \end{aligned}$$

From the formula of $rsu(X,z)$, $rlu(X,z)$, $rreu(X \cup z)$, $RTWU(X \cup z)$, the relationship between the upper bounds can be calculated as follows:

- $rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} u(X,T_k) + u(z,T_k) + rru(z,T_k)$ and $rlu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + rru(X,T_k)]$. $rru(X,T_k) = \sum_{i_j \in T_k \bigwedge i \succ w, \forall w \in X \bigwedge u(i_j,T_k) > 0} u(i_j,\ T_k) \geq u(z,T_k) + rru(z,T_k) = u(X \cup \{z\}) + \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} rru(z,T_k) = \sum_{i_j \in T_k \bigwedge i \succ z \bigwedge u(i_j,T_k)) \rangle 0} u(i_j,\ T_k)$. Let $E(X,T_k) = \{i | i \in T_k \wedge i \succ w, \forall w \in X\}$ and $E(z,T_k) = \{i | i \in T_k \wedge i \succ z\}$, since, $z \in E(X,T_k)$, $\sum_{i_j \in T_k \wedge i \succ w, \forall w \in X \wedge u(i_j,T_k)) \rangle 0} u(i_j,\ T_k) \geq \sum_{i_j \in T_k \wedge i \succ z \bigwedge u(i_j,T_k)) \rangle 0} u(i_j,\ T_k)$. $\boldsymbol{rsu}(X, z) \leq \boldsymbol{rlu}(\boldsymbol{X},z)$ holds.
- $rsu(X,z) = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(X,T_k) + u(z,T_k) + rru(z,T_k)] = \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} [u(\{X \cup z\},T_k) + rru(z,T)] = rreu(X \cup z)$.$\boldsymbol{rsu}(\boldsymbol{X},z) = \boldsymbol{rreu}(\boldsymbol{X} \cup z)$ holds.
- $RTWU(X \cup z) = \sum_{\{X \cup z\} \subseteq T_k \bigwedge T_k \in D} TU(T_k) \geq \sum_{T_k \in D \wedge \{X \cup z\} \subseteq T_k} u(X,T_k) + rru(X,T_k)$. $\boldsymbol{RTWU}(\boldsymbol{X} \cup z) \geq \boldsymbol{rlu}(X,z)$ holds.

For example: based on the information provided in Table 3 and the alphabetical arrangement of the items within, then:

$$RTWU(bd) = 40 > rlu(b,d) = 36 > rsu(b,d) = 31 = rreu(bd) \rangle u(bd) = 25$$

Definition 16. Primary and Secondary items (Singh et al., 2018).

Given an itemset X, the primary items of itemset X is the item set denoted by $Primary(X)$ and calculated by the formula

$$Primary(X) = \{z | z \succ w, \forall w \in X \wedge rsu(X,z) \geq minU\}$$

The Secondary items of itemset X are the itemset denoted by $Secondary(X)$ and calculated by the formula

$$Secondary(X) = \{z | z \succ w, \forall w \in X \wedge rlu(X,z) \geq minU\}$$

Because of $rsu(X,z) \leq rlu(X,z)$, $Primary(X) \subseteq Secondary(X)$.

For example: based on the information provided in Table 3 and the alphabetical arrangement of the items within, for $X = \{bd\}$, table value RSU and RLU of X with each item after X are present in Table 10:

Assuming that $minU = 15$, then $Primary(X) = \{e,f\}$ and $Secondary(X) = \{e,f,g\}$.

Definition 17. Utility Array (Singh et al., 2018).

The Utility Array is defined as an array of length k where k is the number of single items in the database D. The Utility Array is used to store, calculate, and quickly access utility values by passing the $UA[i]$ array in which i is the index of the i^{th} element. Utility Array is defined in (Singh et al., 2018). This structure is used to store the value of $RTWU(X)$, $rlu(X,i)$, $rsu(X,i)$ of item i into the value $UA[i]$ by creating three new corresponding arrays of each value type to be calculated.

4.1. EMHUN algorithm

In order to solve the problem of HUIM in a database with unstable negative profits, this article proposes the EMHUN algorithm, as shown in detail in Algorithm 1. The algorithm is a modified version of the EHIN algorithm (Singh et al., 2018), with the difference found in line #3. The EHIN algorithm only has two lists of items with positive and negative utility in the entire database, while the EMHUN algorithm has an additional list containing hybrid items (which can carry either negative or positive utility in the database). The next difference is encountered in line #7. The EHIN algorithm will sort all positive items on the whole database first and then sort all negative items later. However, the EMHUN algorithm contains three types of items, so the items will be sorted by Definition 7.

The EMHUN algorithm takes as input a database and a $minU$

threshold, and its output is all itemsets whose utility is not less than *minU*. From the database, the algorithm partitions the set of items into three different types of items, including those with positive utility in the entire database (set ρ - line #2), items with negative utility across the entire database (set δ- line #3), and the hybrid items (with either negative or positive utility) (set η - line #4). Then the algorithm scans the database and calculates the RTWU for all items with positive utility and negative utility (set $\rho \cup \delta$) in line #5. The values are calculated based on an array structure called the Utility Array defined in (Singh et al., 2018). Line #6 creates a $Secondary(\varnothing)$ list with items with $RTWU \geq minU$. Line #7 will sort them based on Definition 7. The algorithm scans the database to remove all items that are not in the $Secondary(\varnothing)$ list. EMHUN then removes all non-potential items from the transactions and rearrange all the remaining items into three segments: positive items, hybrid items and negative items. In each segment, the items will be sorted in ascending order of the RTWU value. This is done in line #9.

Algorithm 1: Propsed EMHUN algorithm

```
   Input: Database $D$, threshold $minU$.
   Output: High utility itemsets.
1  $X \leftarrow \varnothing$;
2  $\rho \leftarrow$ set of items having positive utility only in $D$;
3  $\delta \leftarrow$ set of items having both negative and positive utility in $D$;
4  $\eta \leftarrow$ set of items having negative utility only in $D$;
5  Scan $D$ to calculate $RLU(X,i)$ for all item $i \in (\rho \cup \delta)$, using UA;
6  $Secondary(X) = \{i | i \in (\rho \cup \delta) \wedge RLU(X,i) \geq minU\}$;
7  Sort $Secondary(X) \cup \eta$ by definition;
8  Scan D to remove item $x \notin (Secondary(X) \cup \eta)$;
9  Sort the items in the remaining transactions in the order of items with positive
   utility only, items with both negative and positive utility, items with negative
   utility only;
10 Sort transactions in the database $D$;
11 Scan $D$ to compute $RSU(X,i)$, $\forall i \in Secondary(X)$, using UA;
12 $Primary(X) = \{i | i \in Secondary(X) \wedge RSU(X,i) \geq minU\}$;
13 $Search(\eta, X, D, Primary(X), Secondary(X), minU)$;
```

The algorithm further sorts the transactions to facilitate transaction merging using Definition 13, and this is done in line #10. Although EMHUN carried out several sorting methods, they only happened once. In contrast, the proposed algorithm continues to use the Utility Array structure to calculate the $su(\varnothing, i)$ value for each item i in the $Secondary(\varnothing)$, then retrieves all the items with the value $su(\varnothing, i) \geq minU$ to create the primary list. Next, the algorithm recursively performs search space exploration by calling the Search function (Algorithm 2), using inputs as a list of items with negative utility on the whole database, an empty itemset, database D, the primary items and secondary items calculated in lines #6 and #12 and the threshold value *minU*.

The *Search* algorithm takes as inputs a list of items with negative utility across the entire database - η, the itemsets to be explored further X, database D_X, a list of primary items - $Primary(X)$, a list of secondary items - $Secondary(X)$ and a threshold *minU*. The output of the algorithm is that all itemsets are HUI generated by items with non-negative utility across the entire database. For each item i in $Primary(X)$ the algorithm then scans the entire database to create a projected database according to D_β with $\beta = X \cup \{i\}$, This is shown in line #3. In the process of scanning the database to generate D_β, the algorithm also calculates the value of $u(\beta)$ if $u(\beta)$ is not less than *minU*, then β is a HUI, this process is shown from lines #4 to #6. If $u(\beta)$ is greater than the *minU* threshold, then the algorithm will recursively call the *SearchN* algorithm with the input list of items with negative utility on the database - η, itemset β, database D_β, and the *minU* threshold to find all the HUIs that are extended from β and negative utility items across the entire database. This process is shown in lines #7 to #9 in the algorithm. Line #10 will perform a scan of the database D_β to compute the $RSU(\beta, i)$ and $RLU(\beta, i)$ with each item i in the $Secondary(X)$ list and filter out the items that satisfy $Primary(\beta)$ and $Secondary(\beta)$ (lines #11 and #12). The algorithm continues recursively to find HUI with the expansion of β.

Algorithm 2: The Search Procedure

Input: Set of items with negative utility only η, Itemset X, Database D_X, primary items $Primary(X)$, secondary items $Secondary(X)$, threshold $minU$.
Output: High utility itemsets found by appending items to X with items with positive utility only, items with both negative and positive utility.

```
1  foreach $i \in Primary(X)$ do
2      $\beta = X \cup \{i\}$;
3      Scan $D_X$ to calculate $u(\beta)$ and create $D_\beta$
4      if $u(\beta) \geq minU$ then
5          Output $\beta$;
6      end
7      if $u(\beta) \rangle minU$ then
8          $SearchN(\eta, \beta, D_\beta, minU)$;
9      End
10     Scan $D_\beta$ to calculate $RSU(\beta, z), RLU(\beta, z)$ for all $z \in Secondary(X)$ after $i$,
       using two UAs;
11     $Primary(\beta) = \{z | z \in Secondary(X) \wedge RSU(\beta, z) \geq minU\}$;
12     $Secondary(\beta) = \{z | z \in Secondary(X) \wedge RLU(\beta, z) \geq minU\}$;
13     $Search(\eta, \beta, D_\beta, Primary(\beta), Secondary(\beta), minU)$;
14 end
```

The *SearchN* algorithm takes input as a list of all items with negative utility across the entire database - η, itemset X, database D_X, and the *minU* threshold. The algorithm returns HUIs that are extended from X by adding items in η after X. For each item i in list η, the algorithm scans the database to find the database projected with β with $\beta = \{X \cup i\}$. Similar to the Search algorithm, in the process of scanning the database the *SearchN* algorithm (Algorithm 3) will also calculate $u(\beta)$ to check whether $u(\beta)$ is a HUI. The algorithm then scans the database D_β to calculate the RSU of β for each item i in η. The algorithm will filter out the list of $Primary(\beta)$. To discover itemsets extended from β, the Search procedure is called recursively for depth-first search.

Algorithm 3: The SearchN Procedure

Input: Set of items with negative utility only η, Itemset X, Database D_X, threshold $minU$
Output: High utility itemsets found by appending items to X with items with negative utility only

```
1  foreach $i \in \eta$ do
2      $\beta = X \cup \{i\}$;
3      Scan $D_X$ to calculate $u(\beta)$ and create $D_\beta$
4      if $u(\beta) \geq minU$ then
5          Output $\beta$;
6      end
7      Scan $D_\beta$ to calculate $RSU(\beta, z)$ for all $z \in \eta$ after $i$, using two UA;
8      $Primary(\beta) = \{z | z \in \eta \wedge RSU(\beta, z) \geq minU\}$;
10     $SearchN(Primary(\beta), \beta, D_\beta, minU)$;
14 end
```

4.2. A detailed example

To better understand the algorithm, the article will execute a detailed example to illustrate the EMHUN algorithm with the database in Table 3 and $minU = 25$.

- **Step 1:** Assign $X = \varnothing$.
- **Steps 2, 3, 4:** The algorithm will scan the database for three categories of objects: items with positive utility in all of the database - ρ, items with negative and positive utility in the entire database -δ, and items with negative utility in the entire database -η. Following a database scan in Table 3, the values in the three lists include:
    - ρ: c, d, e
    - δ: a, b
    - η: f, g
- **Step 5:** All of the database items' RTWU values have been computed by the algorithm and are shown in Table 11.
- **Step 6:** Since no item has an RTWU less than *minU*, all items are promising and are not pruned from the search space and $Secondary(X) = \{a, b, c, d, e\}$.

Table 11
Items in the database's RTWU.

Items	a	b	c	d	e	f	g
RTWU	29	56	56	49	60	44	25

- **Step 7:** The algorithm then sorts the elements into the order defined in Definition 7, yielding the following result after sorting: $Secondary(X) = \{d, c, e, a, b\}$ and $\eta = \{g, f\}$.
- **Steps 8, 9:** In each database transaction, the item have been sorted.
- **Step 10:** The database transactions are then sorted, and Table 12 shows the results.
- **Step 11:** The method next examines the database in Table 12 to calculate the $RSU(X, i)$ for each item i in the $Secondary(X)$ list. Table 13 summarizes the findings.
- **Step 12:** List of $Primary(X) = \{d, c\}$
- **Step 13:** Algorithm to recursively call Search function with inputs including: list negative $-\eta = \{g, f\}$, itemset X, database D in Table 12, $Primary(X) = \{d, c\}$,$Secondary(X) = \{d, c, e, a, b\}$, *minU*.
- The details of running the recursive Search method include:
  - With each item in $Primary(X) = \{d, c\}$, the algorithm will start with item *"d"*.
  - $\boldsymbol{\beta = \{d\}}$.
  - Table 14 shows the results of the algorithm's exploration of the data in Table 8 to generate D_β.
  - Since $u(d) = 28 > 25$, *"d"* is a HUI and procedure SearchN is called with parameters $\eta = \{g, f\}$, $\beta = \{d\}$, $D_{\{d\}}$, *minU*
  - With each item in $\eta = \{g, f\}$, the algorithm will start with item g
  - $\boldsymbol{\beta = \{dg\}}$.
  - In order to find D_β, the algorithm looks over the data in Table 14; D_β is shown in Table 15.
  - Since $u(dg) = 8$, *"dg"* is not a HUI.
  - Calculating $RSU(\beta, i)$ for each item $i \in \eta$ and after *"g"* in the database, the algorithm's output is shown in Table 16.
  - $Primary(\beta) = \{\}$
  - The algorithm continues with item f
  - $\boldsymbol{\beta = \{df\}}$.
  - Since $u(df) = 18$, *"df"* is not a HUI.
  - The algorithm returns $\boldsymbol{\beta = \{d\}}$**.** The algorithm will traverse the database, calculating the $RSU(i)$ and $RLU(i)$ for each item i after *"d"* in the $Secondary(X)$ list, and the results are shown in Table 17.
  - $Primary(\beta) = \{c, d, b\}$ and $Secondary(\beta) = \{c, d, e, b\}$
  - Recursively call Search with the parameters $\eta = \{g, f\}$,$\beta = \{d\}$, $D_{\{d\}}$, $Primary(\beta) = \{c, d, b\}$, $Secondary(\beta) = \{c, d, e, b\}$, *minU*.

Table 12
The database after sorting the items in the transaction and sorting the transactions.

TID	Items	Profits
T_6	d,c,e,a,b,g,f	$8, 4, 1, 2, 1, -2, -3$
T_1	d,e,a,b,g,f	$4, 3, -4, 2, -2, -2$
T_3	d,c,e,b,f	$12, 1, 2, -2, -1$
T_5	a,f	$4, -3$
T_7	c,e,b	$4, 4, 3$
T_2	c,b	$5, -1$
T_4	d,c,e	$4, 2, 3$

Table 13
The results of the RSU calculation between X and each item in Secondary(X).

Items	d	c	e	a	b
RSU	49	32	21	5	3

Table 14
Database projection results with β where..$\beta = \{a\}$

TID	Items	Profits
T_6	c,e,a,b,g,f	$4, 1, 2, 1, -2, -3$
T_1	e,a,b,g,f	$3, -4, 2, -2, -2$
T_3	c,e,b,f	$1, 2, -2, -1$
T_4	c,e	$2, 3$

Table 15
The results of database projection with β with..$\beta = \{dg\}$

TID	Items	Profits
T_6	f	-3
T_1	f	-2

Table 16
The $RSU(\beta, i)$ values for each i in the following g, where $\beta = \{dg\}$.

Items	f
RSU	8

Table 17
The results of $RSU(\beta, i)$ $RLU(\beta, i)$ for each item i, with..$\beta = \{d\}$

Items	c	d	e	b
RLU	40	49	25	40
RSU	40	42	13	25

The iterative process applies a similar calculation to the remaining itemsets in the search space. After the completion of the algorithm's further exploration of the remaining search space, we obtain the final results (Table 18).

5. Experiments

5.1. Experimental environment and database

5.1.1. Experimental environment

To show the efficiency of the mining performance of the EMHUN algorithm, we evaluate the execution time, memory consumption, the number of generated candidates and the effect of the number of hybrid items on the execution time with the previously proposed EHMIN algorithm. Please note that EHMIN is the newest algorithm proposed in this mining category. However, the EHMIN algorithm does not solve the problem of databases with unstable negative profits. Therefore, we have modified the EHMIN algorithm (Kim et al., 2022) to make it work with databases that include unstable negative profits.

Table 18
All the itemsets that are HUIs in the database.

Itemset	Utility
d	28
dc	31
dce	37
$dceb$	27
de	37
deb	31
$debf$	25
db	5

In order to enhance the EHMIN algorithm to operate effectively with databases featuring variable negative profits, we have implemented several adjustments to the original EHMIN algorithm. These adaptions encompass:

- Expanding the search for hybrid items in the database while scanning, rather than restricting the focus solely to two types of items as stipulated in the initial algorithm.
- Implementing the sorting criteria defined in Definition 7 to generate an ordered list of items. Notably, items with positive utility are consistently given precedence, followed by hybrid items, and concluding with items bearing negative utility.
- Applying the same sorting criteria to organize the items within each transaction, maintaining the specified order

All the tests were done on a Windows 10 machine with a Ryzen 3600 CPU and 24 gigabytes of RAM. Java API was utilized to track both algorithms' runtime and memory consumption.

5.1.2. Characteristics and creation of the databases

Experiments were carried out on the following databases: Accidents, Connect, Chainstore, Chess, Kosarak, Mushroom, Pumsb, BMS, and SUSY. The databases used in the experiment are those generated from the databases used to mine the frequent sets taken from the SPMF library.[1]

However, it is noteworthy that the databases available on SPMF, as well as those from previous sources, predominantly encompass data involving stable positive and negative profits. The presence of databases purely exhibiting unstable negative profits is limited. To address this, we devised a methodology that involves introducing randomness to the profit values of items within databases possessing solely positive profits. This process was executed through the following defined steps:

Initiate by importing the original database that exclusively features positive profits.

2. Segment the item list into three distinct categories at the outset, employing a predefined randomization approach with specified ratios. Each item category is subsequently treated as outlined below:

- For items exclusively associated with positive profits: Retain the original profit values from the source database.
- For items exclusively bearing negative profits: Apply a multiplication factor of (-1) to the original profit, yielding a negative value.
- For items categorized as hybrid items: Introduce a random flag value to dictate the nature of the profit (positive or negative). If the flag signifies a positive value, the original profit is maintained; if negative, the original profit is multiplied by (-1).

These additions serve to elucidate the procedure undertaken to construct a database featuring unstable negative profits, addressing the limitations observed within the datasets available in SPMF.

Table 19 provides a list of the tested databases along with their attributes.

As seen in Table 19, each database has $|D|$ transactions, $|I|$ items, the maximum length of a transaction is T_{max}, T_{avg} denotes the average length of all the transactions in each database, $|P|$ denotes the number of positive only utility items, $|N|$ denotes the number of negative only utility items, $|H|$ denotes the number of hybrid items. The database's features, such as density and sparsity, are shown in the last column. The experimental results are presented below.

5.2. Comparison of EMHUN and EHMIN on databases with unstable negative profits

5.2.1. Analysis of the execution time

The experimental results for runtime are shown in Fig. 1. These show that the execution time decreased as the *minU* threshold increased. In dense databases such as Accident, Chess, Connect, Mushroom, Pumsb and SUSY, the database projection and transaction merge strategy works very well, which reduces the time needed to scan the database significantly. The mining time on dense databases has outstanding performance, and the improvement can be up to ~ 100 times with the Accident database (1,054 s vs. 10.7 s), ~650 times with the Chess database (518 s vs. 0.8 s), ~760 times with the Connect database (416 s vs. 0.55 s), ~30 times with the Mushroom database (48.5 s vs. 1.56 s), and ~ 25 times with the Pumsb database (1,070 s vs. 41 s). For the SUSY database, when tested with *minU* threshold at 100 M and lower, EHMIN cannot be executed due to a memory allocation error. In the sparse database experiments on BMS, Chainstore and Kosarak, the execution speed can be 440 times faster with the BMS database, 1.5 times with the Chainstore database, and 5.0 times with the Kosark database. This enormous performance boost can be ascribed to the separate consideration of the hybrid items, and the tight upper bounds that are introduced to help eliminate many candidates. In large databases such as Kosarak, Chainstore and SUSY, the EMHUN algorithm is also shown to be superior to EHMIN. In particular, when mining the SUSY database, the EHMIN algorithm encountered a memory error, but EMHUN can still exploit lower thresholds.

5.2.2. Analysis of the memory usage

The memory usage results are shown in Fig. 2. The charts show that the trend seen in the execution time comparison also applies to memory usage, where lowering the *minU* results in increased memory consumption. On the same database and with the same threshold, EMHUN requires approximately a quarter to up to a half the memory used in EHMIN in dense databases, and much less in sparse databases. The variation in the amount of memory allocated for EMHUN in the mining process is not high, because the cost to store transactions is almost unchanged due to the strategy of merging them. However, the amount of

Table 19
Databases with unstable negative profits characteristics.

DB	$\|D\|$	$\|I\|$	T_{max}	T_{avg}	$\|P\|$	$\|N\|$	$\|H\|$	**Density**
Accidents	340,183	468	51	33.8	38	32.5	29.5	Dense
BMS	59,602	497	267	2.5	34	34	32	Sparse
Connect	67,557	129	43	43	34	32.5	33.5	Dense
Chainstore	1,112,949	46,086	170	7.20	39	34	27	Sparse
Chess	3,196	75	37	37	34.5	32	33.5	Dense
Kosarak	990,000	41,270	2,498	8.1	40.5	34.5	25	Sparse
Mushroom	8,416	119	23	23	34.5	33.5	32	Dense
Pumsb	49,046	2,113	74	74	38	34	28	Dense
SUSY	5,000,000	180	19	19	34	33	33	Dense

[1] https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

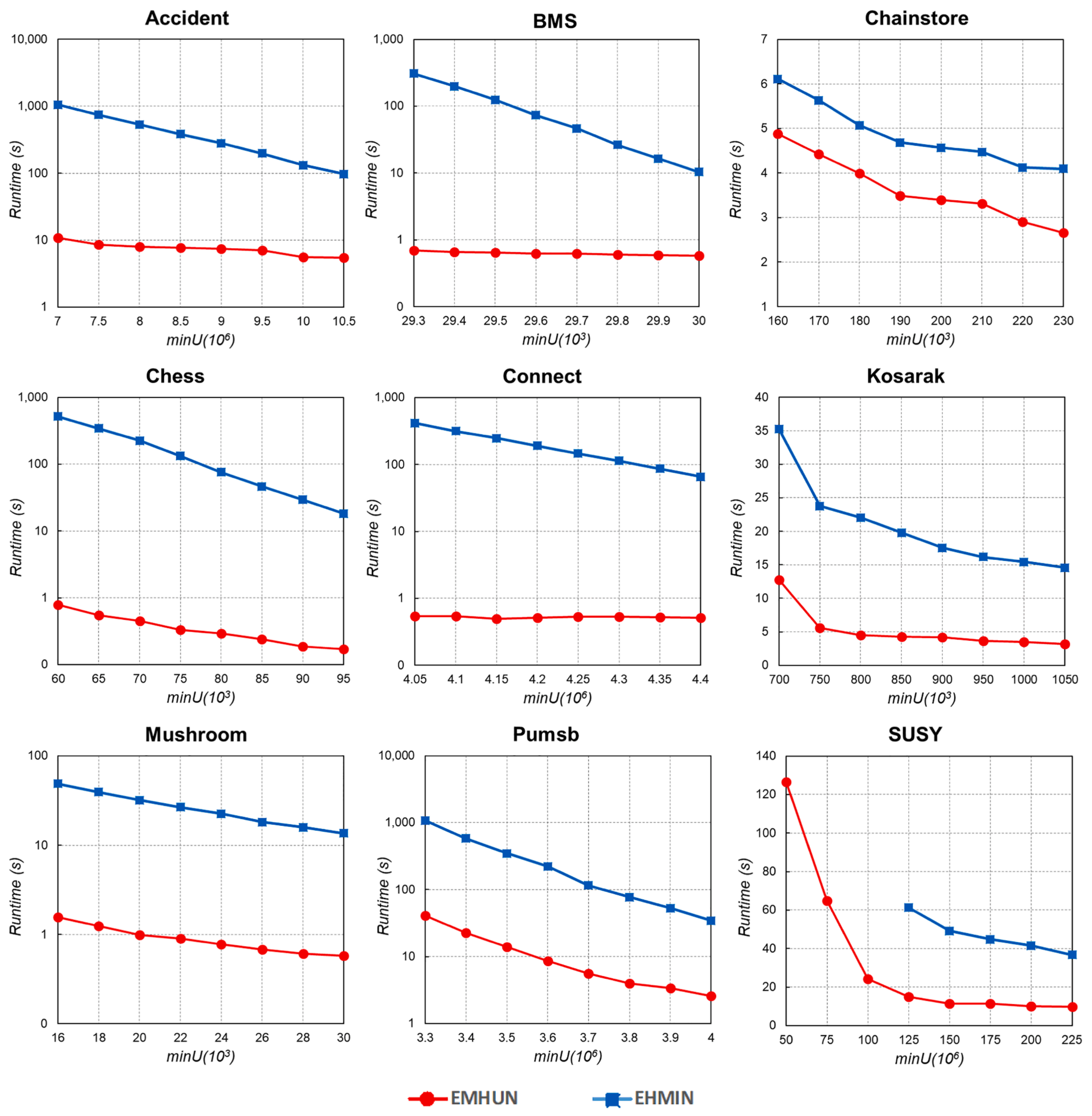

Fig. 1. Comparison of the execution time across various databases with unstable negative profits.

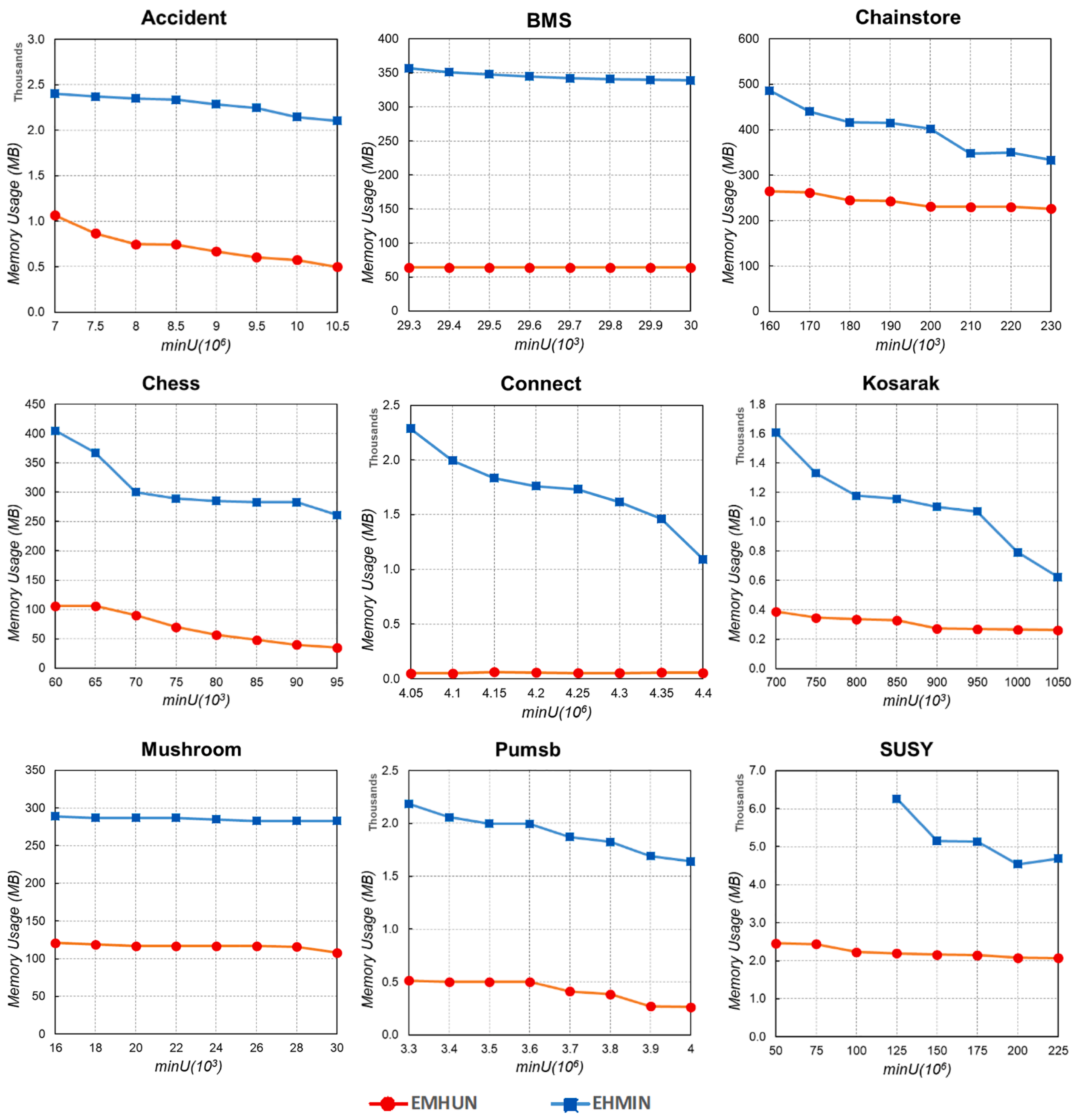

Fig. 2. Comparison of the memory usage across various databases with unstable negative profits.

memory allocated for EHMIN tends to spike at low *minU* thresholds.

5.2.3. Analysis of the number of candidates generated

The number of candidates produced during the testing on various databases is shown in Fig. 3. In most databases, the EHMIN algorithm generates more candidates than our proposed EMHUN algorithm. The difference in the number of candidates produced with databases like Accident and Mushroom is 1.5 times, with Kosarak it is roughly 10 times, with Pumsb it is approximately 30 times, and it can be up to nearly 60 times with Chess or even 5000 times with Connect. The key to reducing the number of candidates is that EMHUN uses an upper threshold of the redefined sub-tree and redefined local utility to prune more candidates, in addition, splitting the list of item types also contributes to reducing the number of candidates generated. In Chainstore and SUSY, although the number of candidates generated by EMHUN is higher than with EHMIN because the number of transactions to be scanned is significantly reduced by the database pooling method, the mining time in these databases also shows the greater efficiency of the proposed algorithm.

5.2.4. Analysis of the effect of the number of hybrid items on the execution time

To comprehensively assess the influence of varying hybrid element counts on execution time, we adopted the following methodology:

- Maintain a consistent minU threshold for all executions, leveraging the same database.
- Maintain a fixed 40% ratio for items featuring solely positive profits.
- Manipulate the ratio of hybrid items, ranging from 10% to 50%, with the remaining comprising items with negative profit only.

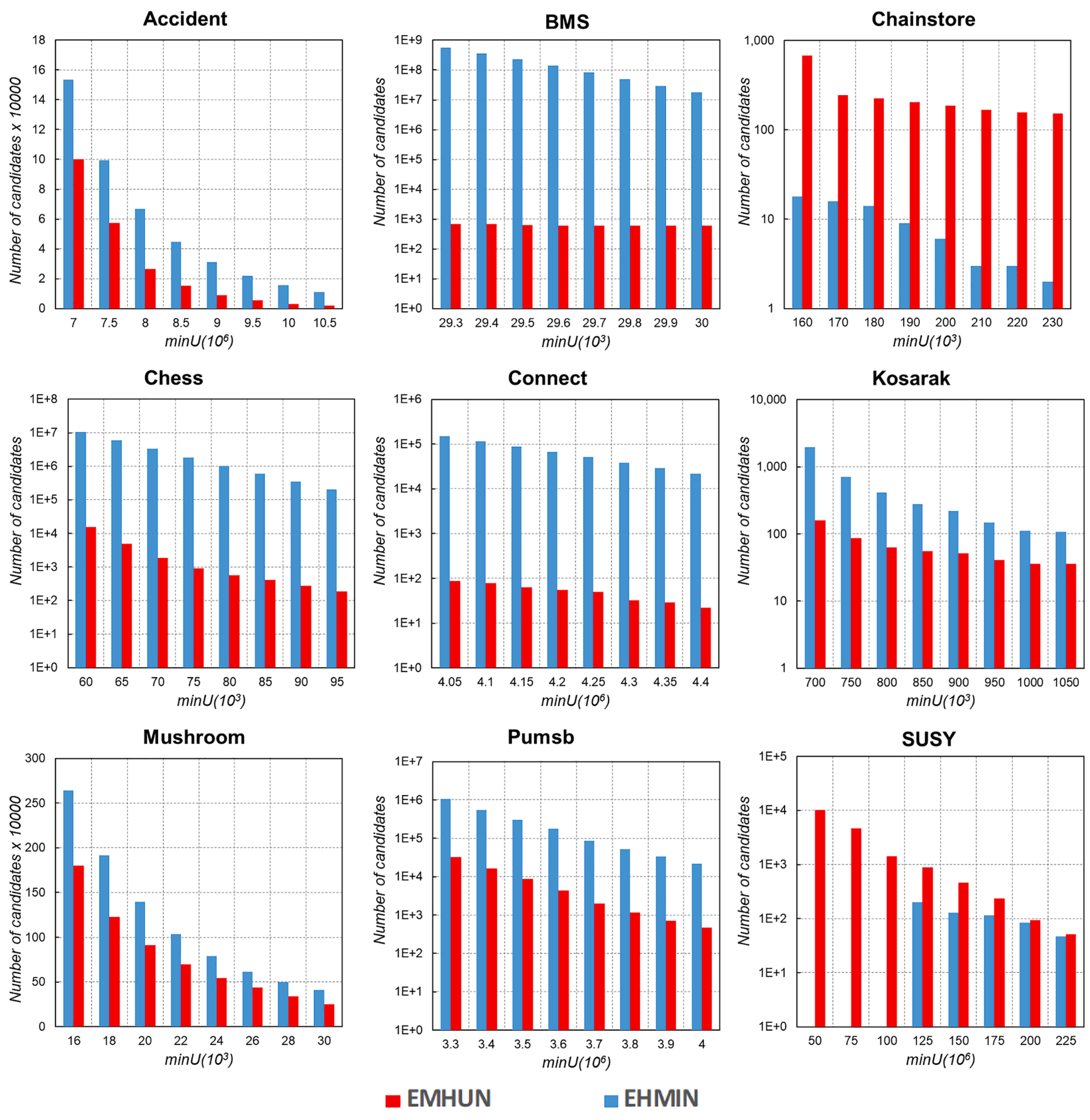

Fig. 3. Comparison of the number of candidates across various databases with unstable negative profits.

Record and present the execution times for each run alongside the corresponding database through the charts in Fig. 4.

Analyzing the outcomes depicted in the charts, the following observations are evident: An increase in the number of hybrid items leads to a corresponding increase in consumed time. This phenomenon arises from the reduction in items with exclusively negative profits, subsequently elevating itemset utility. Consequently, this leads to an expansion in candidate counts, consequently amplifying algorithmic time consumption.

For databases characterized by high density, such as Chess, Connect, Mushroom, Pumsb, and SUSY, the change in hybrid item ratios doesn't significantly impact execution time for the EMHUN algorithm. However, the EHMIN algorithm exhibits substantial variance in time consumption. In the Chess database, the execution time for EMHUN increased from 0.1 s to 0.7 s, whereas EHMIN escalated from 0.4 s to 113 s. Similar trends are observed in the Connect database (from 0.3 s to 0.6 s for EMHUN, compared to from 1.8 s to 231 s for EHMIN), the Mushroom database (from 1.9 s to 3.6 s for EMHUN, compared to from 10 s to 73 s for EHMIN), and the Pumsb database (from 0.2 s to 0.8 s for EMHUN, compared to from 3.2 s to 195 s for EHMIN). The EMHUN algorithm's transaction merging strategy accounts for the minimal shift in execution time, as the change in candidates barely affects the cost of database scanning. Conversely, EHMIN's promising candidate growth with an increasing number of hybrid items accentuates execution time.

Sparse databases exhibit similar trends; however, EMHUN's progressions are still notably quicker than EHMIN. For instance, in the Accidents database, EMHUN's execution time rose from 3.7 s to 13.5 s, while EHMIN's surged from 12.3 s to 63.5 s. Similarly, in the Kosarak database, EMHUN's time increased from 5.8 s to 10.7 s, while EHMIN's expanded from 6.3 s to 13.5 s. The incorporation of redefined subtree utility and redefined local utility in the EMHUN algorithm enhances early pruning, effectively reducing time-consuming candidate mining.

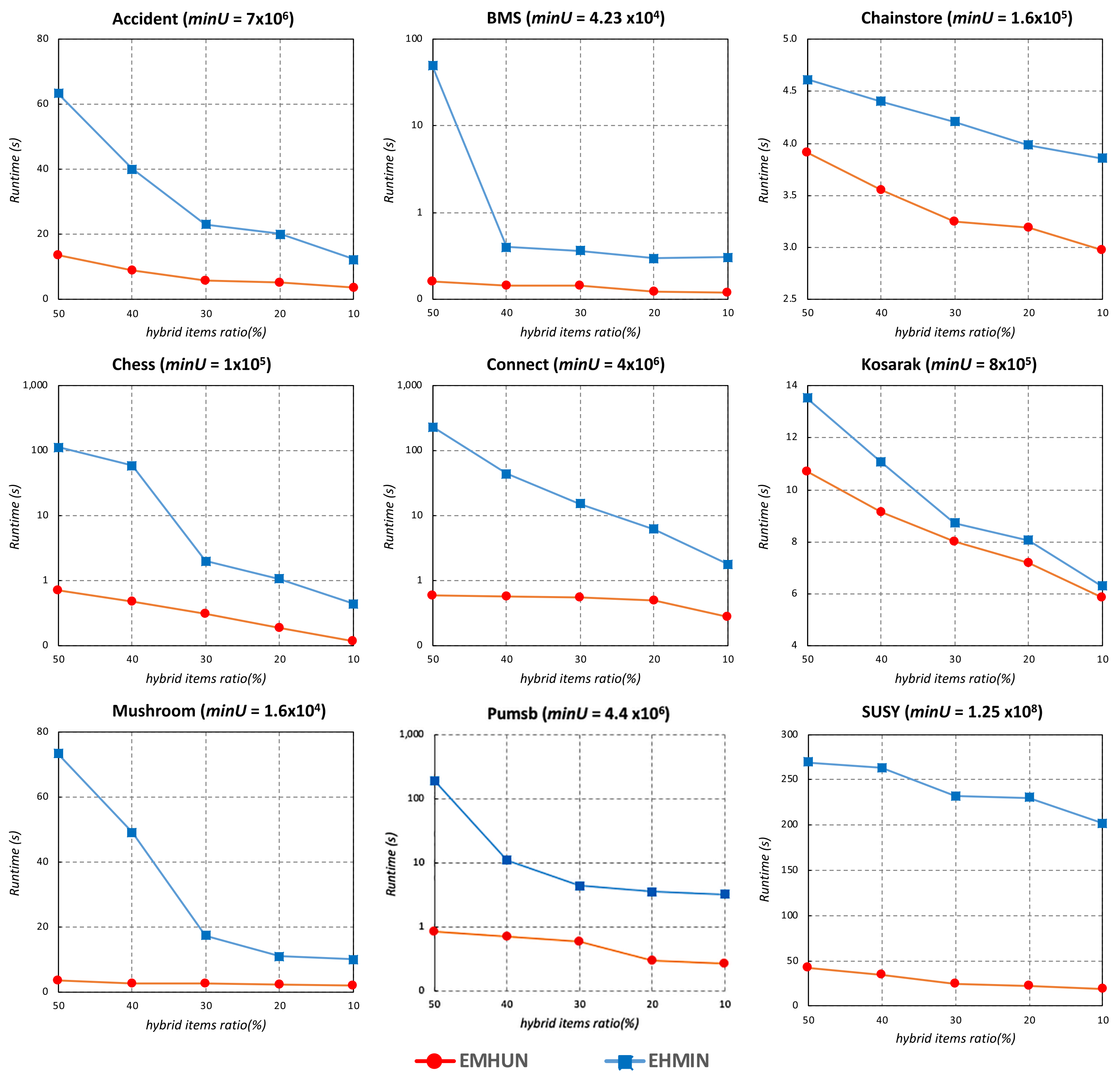

Fig. 4. Comparison of the effect of the number of hybrid items on the execution time across various databases.

Overall, across all databases, the EMHUN algorithm consistently outperforms EHMIN in terms of execution time. Even with fluctuations in hybrid item numbers, the EMHUN algorithm consistently demonstrates superior adaptability regarding execution time.

6. Conclusion and future works

This study analyzed both the theoretical and practical aspects of unstable negative profits databases. To effectively extract the high utility itemsets from this data, this paper developed the EMHUN algorithm. The study employs efficient ways to reduce the search space by redefining upper bounds, such as the redefined sub-tree utility and redefined local utility, to solve the suggested issue. Furthermore, the correctness of the results when using these upper bounds is also proven. The mining process is also optimized by classifying hybrid items into a new list and having a separate treatment for this list, leading to a great improvement in mining speed. Finally, the strategy of transaction consolidation to reduce the number of transactions that need to be scanned is also incorporated into the algorithm. Experiments with the EHMIN algorithm show that the improvement in speed can be up to more than 750 times in dense databases or 440 times in sparse databases. In addition, MEHUN can still be used at smaller thresholds in large databases, while EHMIN cannot. Experiments also demonstrate that when changing the number of hybrid items, MEHUN is less affected compared to the most modern algorithm today, EHMIN. The execution time of MEHUN shows much less variation than that of EHMIN.

Databases with unstable negative utilities will be the focus of future work, including closed HUIM, maximal HUIM, and top-k HUIs. In addition, we are always looking at new approaches for parallel or distributed execution to boost mining efficiency and meet practical needs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Ahmed, U., Lin, J.-C.-W., Srivastava, G., Yasin, R., & Djenouri, Y. (2021). An evolutionary model to mine high expected utility patterns from uncertain databases. *IEEE Transactions on Emerging Topics in Computational Intelligence, 5*(1), 19–28. https://doi.org/10.1109/TETCI.2020.3000224

Ashraf, M., Abdelkader, T., Rady, S., & Gharib, T. F. (2022). TKN: an efficient approach for discovering top-k high utility itemsets with positive or negative profits. *Information Sciences, 587*, 654–678. https://doi.org/10.1016/j.ins.2021.12.024

Chu, C.-J., Tseng, V. S., & Liang, T. (2009). An efficient algorithm for mining high utility itemsets with negative item values in large databases. *Applied Mathematics and Computation, 215*(2), 767–778. https://doi.org/10.1016/j.amc.2009.05.066

Dam, T. L., Li, K., Fournier-Viger, P., & Duong, Q. H. (2019). CLS-Miner: Efficient and effective closed high-utility itemset mining. *Frontiers of Computer Science, 13*(2), 357–381.

Duong, H., Hoang, T., Tran, T., Truong, T., Le, B., & Fournier-Viger, P. (2022). Efficient algorithms for mining closed and maximal high utility itemsets. *Knowledge-Based Systems, 257*, Article 109921. https://doi.org/10.1016/j.knosys.2022.109921

Duong, Q.-H.-H., Fournier-Viger, P., Ramampiaro, H., Nørvåg, K., & Dam, T.-L.-L. (2018). Efficient high utility itemset mining using buffered utility-lists. *Applied Intelligence, 48*(7), 1859–1877. https://doi.org/10.1007/s10489-017-1057-2

Fournier-Viger, P., Wu, C.-W., Zida, S., & Tseng, V. S. (2014). FHM: Faster High-Utility itemset mining using estimated utility co-occurrence pruning. In *In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 8502 LNAI* (pp. 83–92). https://doi.org/10.1007/978-3-319-08326-1_9

Gan, W., Lin, J.-C.-W., Fournier-Viger, P., Chao, H.-C., Tseng, V. S., & Yu, P. S. (2021). A survey of Utility-Oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering, 33*(4), 1306–1327. https://doi.org/10.1109/TKDE.2019.2942594

Gan, W., Lin, J.-C.-W., Fournier-Viger, P., Chao, H.-C., & Yu, P. S. (2019). A survey of parallel sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data, 13*(3), 1–34. https://doi.org/10.1145/3314107

Gan, W., Lin, J.-C.-W., Fournier-Viger, P., Chao, H.-C., & Yu, P. S. (2020). HUOPM: High-Utility occupancy pattern mining. *IEEE Transactions on Cybernetics, 50*(3), 1195–1208. https://doi.org/10.1109/TCYB.2019.2896267

Han, X., Liu, X., Li, J., & Gao, H. (2021). Efficient top-k high utility itemset mining on massive data. *Information Sciences, 557*, 382–406. https://doi.org/10.1016/j.ins.2020.08.028

Kim, H., Ryu, T., Lee, C., Kim, H., Yoon, E., Vo, B., Chun-Wei Lin, J., & Yun, U. (2022). EHMIN: Efficient approach of list based high-utility pattern mining with negative unit profits. *Expert Systems with Applications*, *209*(July 2021), 118214. https://doi.org/10.1016/j.eswa.2022.118214.

Krishnamoorthy, S. (2015). Pruning strategies for mining high utility itemsets. *Expert Systems with Applications, 42*(5), 2371–2381. https://doi.org/10.1016/j.eswa.2014.11.001

Krishnamoorthy, S. (2017a). Efficiently mining high utility itemsets with negative unit profits. *Knowledge-Based Systems, 145*. https://doi.org/10.1016/j.knosys.2017.12.035

Krishnamoorthy, S. (2017b). HMiner: Efficiently mining high utility itemsets. *Expert Systems with Applications, 90*, 168–183. https://doi.org/10.1016/j.eswa.2017.08.028

Krishnamoorthy, S. (2019). Mining top-k high utility itemsets with effective threshold raising strategies. *Expert Systems with Applications, 117*, 148–165. https://doi.org/10.1016/j.eswa.2018.09.051

Le, B., Nguyen, H., & Vo, B. (2011). An efficient strategy for mining high utility itemsets. *International Journal of Intelligent Information and Database Systems, 5*(2), 164–176.

Lee, J., Yun, U., Lee, G., & Yoon, E. (2018). Efficient incremental high utility pattern mining based on pre-large concept. *Engineering Applications of Artificial Intelligence, 72*, 111–123. https://doi.org/10.1016/j.engappai.2018.03.020

Lin, J.-C.-W., Fournier-Viger, P., & Gan, W. (2016). FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits. *Knowledge-Based Systems, 111*, 283–298. https://doi.org/10.1016/j.knosys.2016.08.022

Liu, J., Wang, K., & Fung, B. C. M. (2012). Direct discovery of high utility itemsets without candidate generation. In *Proceedings - IEEE International Conference on Data Mining* (pp. 984–989).

Liu, M., & Qu, J. (2012). Mining high utility itemsets without candidate generation. *Proceedings of the 21st ACM International Conference on Information and Knowledge Management - CIKM '12*, 55. https://doi.org/10.1145/2396761.2396773.

Y. Liu W.K. Liao A. Choudhary A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets 2005 Springer-Verlag 689 695 10.1007/11430919_79.

Luna, J. M., Fournier-Viger, P., & Ventura, S. (2019). Frequent itemset mining: A 25 years review. *WIREs. Data Mining and Knowledge Discovery, 9*(6). https://doi.org/10.1002/widm.1329

Luna, J. M., Kiran, R. U., Fournier-Viger, P., & Ventura, S. (2023). Efficient mining of top-k high utility itemsets through genetic algorithms. *Information Sciences, 624*, 529–553. https://doi.org/10.1016/j.ins.2022.12.092

Mai, T., Nguyen, L. T. T., Vo, B., Yun, U., & Hong, T.-P. (2020). Efficient algorithm for mining Non-Redundant High-Utility association rules. *Sensors, 20*(4), 1078. https://doi.org/10.3390/s20041078

Nguyen, L. T. T., Nguyen, P., Nguyen, T. D. D., Vo, B., Fournier-Viger, P., & Tseng, V. S. (2019). Mining high-utility itemsets in dynamic profit databases. *Knowledge-Based Systems, 175*, 130–144. https://doi.org/10.1016/j.knosys.2019.03.022

Nguyen, L. T. T., Vu, D.-B., Nguyen, T. D. D., & Vo, B. (2020). Mining maximal high utility itemsets on dynamic profit databases. *Cybernetics and Systems, 51*(2), 140–160. https://doi.org/10.1080/01969722.2019.1705549

Noaman, A. Y., Luna, J. M., Ragab, A. H. M., & Ventura, S. (2016). Recommending degree studies according to students' attitudes in high school by means of subgroup discovery. *International Journal of Computational Intelligence Systems, 9*(6), 1101. https://doi.org/10.1080/18756891.2016.1256573

Qu, J.-F., Fournier-Viger, P., Liu, M., Hang, B., & Hu, C. (2023). Mining high utility itemsets using prefix trees and utility vectors. *IEEE Transactions on Knowledge and Data Engineering, 1–14*. https://doi.org/10.1109/TKDE.2023.3256126

Ryu, T., Kim, H., Lee, C., Kim, H., Vo, B., Lin, J.-C.-W., … Yun, U. (2022). Scalable and efficient approach for high temporal fuzzy utility pattern mining. *IEEE Transactions on Cybernetics, 1–14*. https://doi.org/10.1109/TCYB.2022.3198661

Sahoo, J., Das, A. K., & Goswami, A. (2015). An efficient approach for mining association rules from high utility itemsets. *Expert Systems with Applications, 42*(13), 5754–5778. https://doi.org/10.1016/j.eswa.2015.02.051

Singh, K., Kumar, A., Singh, S. S., Shakya, H. K., & Biswas, B. (2019). EHNL: an efficient algorithm for mining high utility itemsets with negative utility value and length constraints. *Information Sciences, 484*, 44–70. https://doi.org/10.1016/j.ins.2019.01.056

Singh, K., Shakya, H. K., Singh, A., & Biswas, B. (2018). Mining of high-utility itemsets with negative utility. *Expert Systems, 35*(6), e12296.

Srivastava, G., Lin, J.-C.-W., Pirouz, M., Li, Y., & Yun, U. (2021). A Pre-Large Weighted-Fusion system of sensed High-Utility patterns. *IEEE Sensors Journal, 21*(14), 15626–15634. https://doi.org/10.1109/JSEN.2020.2991045

Subramanian, K., & Kandhasamy, P. (2015). UP-GNIV: An expeditious high utility pattern mining algorithm for itemsets with negative utility values. *International Journal of Information Technology and Management, 14*(1), 26–42. https://doi.org/10.1504/IJITM.2015.066056

Sun, R., Han, M., Zhang, C., Shen, M., & Du, S. (2021). Mining of top-k high utility itemsets with negative utility. *Journal of Intelligent and Fuzzy Systems, 40*(3), 5637–5652. https://doi.org/10.3233/JIFS-201357

Tian, Y., Lu, C., Zhang, X., Cheng, F., & Jin, Y. (2022). A pattern Mining-Based evolutionary algorithm for Large-Scale sparse multiobjective optimization problems. *IEEE Transactions on Cybernetics, 52*(7), 6784–6797. https://doi.org/10.1109/TCYB.2020.3041325

Tseng, V. S., Wu, C.-W., Fournier-Viger, P., & Yu, P. S. (2016). Efficient algorithms for mining Top-K high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering, 28*(1), 54–67. https://doi.org/10.1109/TKDE.2015.2458860

Tseng, V. S., Wu, C. W., Fournier-Viger, P., & Yu, P. S. (2015). Efficient algorithms for mining the concise and lossless representation of high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering, 27*(3), 726–739.

Tseng, V. S., Wu, C. W., Shie, B. E., & Yu, P. S. (2010). UP-Growth: An efficient algorithm for high utility itemset mining. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 253–262. https://doi.org/10.1145/1835804.1835839.

Tung, N. T., Nguyen, L. T. T., Nguyen, T. D. D., Fourier-Viger, P., Nguyen, N.-T., & Vo, B. (2022a). Efficient mining of cross-level high-utility itemsets in taxonomy quantitative databases. *Information Sciences, 587*, 41–62. https://doi.org/10.1016/j.ins.2021.12.017

Tung, N. T., Nguyen, L. T. T., Nguyen, T. D. D., & Vo, B. (2022b). An efficient method for mining multi-level high utility itemsets. *Applied Intelligence, 52*(5), 5475–5496. https://doi.org/10.1007/s10489-021-02681-z

Vo, B., Nguyen, L. T. T., Bui, N., Nguyen, T. D. D., Huynh, V.-N., & Hong, T.-P. (2020). An efficient method for mining closed potential High-Utility itemsets. *IEEE Access, 8*, 31813–31822. https://doi.org/10.1109/ACCESS.2020.2974104

Wu, P., Niu, X., Fournier-Viger, P., Huang, C., & Wang, B. (2022). UBP-Miner: An efficient bit based high utility itemset mining algorithm. *Knowledge-Based Systems, 248*, Article 108865. https://doi.org/10.1016/j.knosys.2022.108865

Xu, T., Dong, X., Xu, J., & Dong, X. (2017). Mining high utility sequential patterns with negative item values. *International Journal of Pattern Recognition and Artificial Intelligence, 31*(10), 1750035. https://doi.org/10.1142/S0218001417500355

Yun, U., Nam, H., Lee, G., & Yoon, E. (2019). Efficient approach for incremental high utility pattern mining with indexed list structure. *Future Generation Computer Systems, 95*, 221–239. https://doi.org/10.1016/j.future.2018.12.029

Zida, S., Fournier-Viger, P., Lin, J.-C.-W., Wu, C.-W., & Tseng, V. S. (2017). EFIM: A fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems, 51*(2), 595–625. https://doi.org/10.1007/s10115-016-0986-0