Engineering Applications of Artificial Intelligence 123 (2023) 106182

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Mining periodic high-utility itemsets with both positive and negative utilities

Fuyin Lai [a], Xiaojie Zhang [a], Guoting Chen [a,*], Wensheng Gan [b,*]

[a] School of Science, Harbin Institute of Technology (Shenzhen), Shenzhen, China
[b] College of Cyber Security, Jinan University, Guangzhou, China

ARTICLE INFO

Keywords:
Periodic pattern
High-utility pattern
Upper bound
Negative utilities

ABSTRACT

Mining high-utility patterns in databases containing items with both positive and negative profits is useful in market basket databases, since negative profits are common in the real world. Obviously, in the market basket database, patterns with stable long-term profits have more meaning. The discovery of itemsets with a consistent high frequency is known as periodic frequent pattern mining. Therefore, mining periodic high-utility patterns in a database containing items with both positive and negative profits is an interesting and useful task. However, this task has two main challenges. First, the utility measure does not have the download closure property. Second, the huge search space needs to be pruned more effectively. In this paper, we propose a vertical data structure-based algorithm called PHMN to discover periodic high-utility patterns (PHUPs) or itemsets in a transaction database with both positive and negative utilities. To be more efficient, we propose a new upper bound to prune the search space and an improved algorithm to discover the PHUPs. Finally, experiments are conducted to verify the effectiveness and efficiency of algorithms.

1. Introduction

Pattern mining (Fournier-Viger et al., 2022; Gan et al., 2019) has been proven to be a useful tool in different application scenarios. Many algorithms are proposed to discover itemsets, which can provide suggestions for decision-makers. Apriori (Agrawal et al., 1994) and FP-Growth (Han et al., 2004) are two classic algorithms for frequent pattern mining. Apriori (Agrawal et al., 1994) uses a lever-wise combination to generate candidate itemsets, and FP-Growth (Han et al., 2004) uses a pattern tree to generate candidate itemsets. Apriori is a two-phase algorithm, which means it needs to find candidates first and then verify whether they are highly frequent patterns. Generally, two-phase algorithms are less efficient since they need to traverse a database many times to generate candidates and have a time-consuming verifying process. Zaki (2000) proposed a one-phase algorithm to discover frequent patterns by storing information in list structures, and the efficiency of the algorithm is greatly improved compared with previous algorithms (Agrawal et al., 1994; Han et al., 2004). In 2019, Luna et al. (2019) reviewed frequent itemset mining in recent years.

However, in frequent pattern mining, the utilities of items are ignored. For example, we have an "apple" in a transaction database. We only consider whether "apple" appears in a transaction, but we do not consider how many "apple" appear in a transaction. The loss of information makes it difficult to dig out patterns that are truly instructive. Consequently, utility mining (Chan et al., 2003; Gan et al., 2021, 2018) is proposed, which is more difficult than frequent pattern mining, because the utility does not satisfy the downward closure property, which makes the search space unable to get effectively pruned. Many new algorithms are designed to discover high-utility patterns efficiently (Liu and Qu, 2012; Tseng et al., 2010). Utility mining has been applied in cross-marketing, biomedical data analysis, stream analysis, and other fields (Erwin et al., 2008; Gan et al., 2021; Li et al., 2011).

Mining meaningful patterns in databases that are closer to reality is an eternal topic. In real databases, especially market databases, items with negative profits are very common, which is easy to understand. In general, in order to increase turnover, supermarkets will carry mixed bundling, and the price of the mixed bundling is generally lower than the price of the two products purchased separately. In this case, if we look at each product separately, one product may show a loss, while the other may show a profit. It is a successful marketing strategy when the overall benefit is greater than the benefit brought by separating the two products. Therefore, considering items with both positive and negative utilities is meaningful, but algorithms based on databases without negative utilities cannot be used directly. Up to now, several algorithms (e.g., FHN Lin et al., 2016, HUPNU Gan et al., 2020a, TopHUI Gan et al., 2020c) have been designed to discover high-utility patterns in databases with both positive and negative utilities.

Considering only the utility in a database containing both positive and negative utilities is insufficient. The task of mining patterns with long-term stability and high utility is more interesting. For example, many products, such as many novelty products, take advantage of people's curiosity about the unknown. Normally, these kinds of products will bring a lot of profit in the beginning but will not bring profit

* Corresponding authors.
E-mail addresses: fylai123@163.com (F. Lai), xjzhang2021@163.com (X. Zhang), chenguoting@hit.edu.cn (G. Chen), wsgan001@gmail.com (W. Gan).

https://doi.org/10.1016/j.engappai.2023.106182
Received 2 August 2022; Received in revised form 13 January 2023; Accepted 19 March 2023
Available online 28 March 2023
0952-1976/© 2023 Published by Elsevier Ltd.

for a long period of time. That is to say, these kinds of products may have high utility in our database, but they may not be meaningful because their future value is low. Therefore, discovering patterns that will generate profits for a long time is important. Periodic frequent pattern mining is important for understanding regular and repeating behaviors (Tanbeer et al., 2009). Generally, a pattern that satisfies *minPer* and *maxPer* will be considered a periodic pattern. Compared with frequent patterns, periodic frequent patterns are more stable since they have periodicity. Therefore, if we combine periodicity with utility mining in a database with both positive and negative utilities, the patterns we mine will be more meaningful. However, there is no research on mining for periodic high-utility patterns in databases with both positive and negative utilities.

Consequently, in this paper, we aim to discover periodic high-utility patterns in a database with both positive and negative utilities. The contributions of this paper are as follows:

- We first propose an algorithm, called PHMN, to discover periodic high-utility patterns in a database with both positive and negative utilities, which is meaningful and has not been studied before.
- Then we propose a novel upper bound to prune search space more efficiently and an improved algorithm, PHMN+, based on the new upper bound.
- Experiments are designed to show the effectiveness and efficiency of the proposed algorithms and the novel pruning strategies.

The rest of this paper is organized as follows. Sections 2, 3, 4, and 5 respectively represent related work, some definitions related to periodic high-utility patterns, the PHMN and PHMN+ algorithms, and experiment evaluation in detail. Section 6 gives the conclusion.

2. Related work

2.1. High-utility pattern mining

Frequent pattern mining (FPM) mainly utilizes the measure of supports of itemsets in a binary database (Agrawal et al., 1994; Gan et al., 2017; Han et al., 2004). However, regardless of the quantity and utility of items, giving all items the same weight is obviously inconsistent with the reality of real-life situations. To solve the problem, many studies consider the quantities and utilities of patterns. In 2003, Chan et al. (2003) proposed the concept of high-utility pattern mining (HUPM), and designed a lever-wise itemset mining algorithm to discover top-k high-utility patterns. Liu et al. (2005) proposed a classic two-phase algorithm based on Apriori (Agrawal et al., 1994) to discover high-utility patterns and proposed the concept of TWU (Transaction-weighted Utilization) to prune useless search space, which is widely used in later studies. Inspired by FP-Growth (Han et al., 2004), many algorithms based on tree structure are proposed (Ahmed et al., 2009; Tseng et al., 2012, 2010). Ahmed et al. (2009) proposed a tree-based algorithm, named IHUP, to discover high-utility patterns. In phase 1, IHUP generates candidates by IHUP-tree; and in phase 2, it identifies high-utility patterns. To make the algorithm more efficient, Tseng et al. (2010) proposed the UP-Growth algorithm. The UP-tree and four strategies that can be applied to the UP-Growth algorithm are proposed. Compared with UP-Growth, Tseng et al. (2012) proposed an enhanced algorithm named UP-Growth+ to discover high-utility patterns more efficiently. Liu and Qu (2012) proposed a one-phase algorithm to discover high-utility patterns (HUP), called HUI-Miner. Correspondingly, it utilized the utility-list and stored all information in the list, which avoided multiple scans of the database. Thus, HUI-Miner had great improvements in execution time compared with most two-phase algorithms. Fournier-Viger et al. (2014) proposed a novel pruning strategy, named EUCP (Estimated Utility Co-occurrence Pruning), to improve HUI-Miner, and it is up to six times faster than HUI-Miner. Zida et al. (2017) proposed the EFIM algorithm to discover high-utility patterns. It combines projected database and transaction merging into HUPM, and gets a huge improvement in execution time. The UFH algorithm (Dawar et al., 2017), a hybrid framework to efficiently mine high-utility patterns, can harness the benefits of both a tree-based algorithm and an inverted-list-based algorithm. UFH has a huge improvement compared with HUI-Miner, FHM, and EFIM in some datasets.

In addition to the improvement of mining efficiency, many HUPM algorithms also combine other measures to make patterns more interesting (Baek et al., 2021; Lin et al., 2015a,b; Song et al., 2021; Tung et al., 2022). For example, the mining of periodic high-utility patterns in transaction databases (Fournier-Viger et al., 2016); the task of high-utility occupancy pattern mining (Chen et al., 2021; Gan et al., 2020b); the problem of mining high-utility temporal patterns from time-interval databases, where items will not appear at a timestamp like in a transaction database, but will continue for a period of time (Wang et al., 2020). Nguyen et al. (2021) proposed the closed high-utility patterns, which is a concise representation of high-utility patterns. FHUQI (Nouioua et al., 2021) can discover high-utility quantitative patterns and provide information about the purchase quantities of items in patterns, which is not shown in HUPM.

2.2. Periodic frequent pattern mining

Normally, frequent pattern mining in transaction databases will return many itemsets satisfying the support conditions. In this case, mining more interesting patterns from transaction databases becomes a priority. Periodic frequent pattern mining (PFPM) is a topic of pattern mining, and it is of great importance for understanding regular and repeating behaviors. Tanbeer et al. (2009) proposed a concept of periodic frequent patterns (PFP). It defined patterns satisfying *minSup* and *maxPer* as periodic frequent patterns and proposed a pattern-growth algorithm to discover patterns with a PF-tree structure. Amphawan et al. (2009) proposed a top-k list, which can be used to generate candidate patterns, and proposed the MTKPP algorithm to discover top-k periodic frequent patterns without user-specified support. A new concept of approximate periodicity (Amphawan et al., 2010) was proposed to discover PFP with variations of the PFP-tree and the ITL-tree, which is an efficient algorithm. There is also an algorithm to discover PFP with a greedy approach (Kiran and Kitsuregawa, 2014). These algorithms mine PFP based on a single *minSup* (a given threshold by users, which limits the minimum support of patterns) and *maxPer* (a given threshold by users, which limits the maximum interval of two successive occurrences of a pattern). However, periodic patterns in true databases usually contain both frequent and rare items. To consider PFP containing rare items, the study (Kiran and Reddy, 2009) proposed the multiple minimum support and extended the multiple minimum support and multiple maximum periodicities (Uday Kiran and Krishna Reddy, 2010). Besides, Rashid et al. (2012) proposed a similar concept with periodic frequent patterns, namely regular frequent patterns, which can be mined based on user-specified *minSup* and *maxVar* (maximum variance). Likhitha et al. (2020) considered closed PFP, which is a concise representation of PFP.

However, these periodic frequent patterns need to exhibit complete cyclic repetitions in a database, which is not common because the real world is imperfect. Therefore, partial periodic frequent patterns are proposed to fix the problem, and the former patterns are called full periodic frequent patterns. Kiran et al. (2017) proposed the GPF-Growth algorithm to find both full and partial periodic frequent patterns. GPF-Growth also utilizes a new measure, namely *periodic-ratio*, to determine the interestingness of partial periodic and frequent patterns. A depth-first algorithm called PPF-DFS (Nakamura et al., 2021) was proposed to discover partial periodic frequent patterns efficiently. Note that PPF-DFS is up to 2 to 8.8 times faster than GPF-Growth (Kiran et al., 2017).

Table 1
Database.

Tid	Item	Quantity
T_1	$a\ b\ c\ d$	5 2 1 2
T_2	$a\ c\ d\ g$	1 1 1 3
T_3	$a\ c\ f$	1 1 1
T_4	$a\ f\ g$	1 4 2
T_5	$a\ g$	1 2
T_6	$b\ c\ d\ e$	3 2 3 1
T_7	$c\ e$	6 4
T_8	$e\ f$	1 3

Table 2
Unit utility.

Item	a	b	c	d	e	f	g
Utility	3	6	−3	12	−5	−2	−1

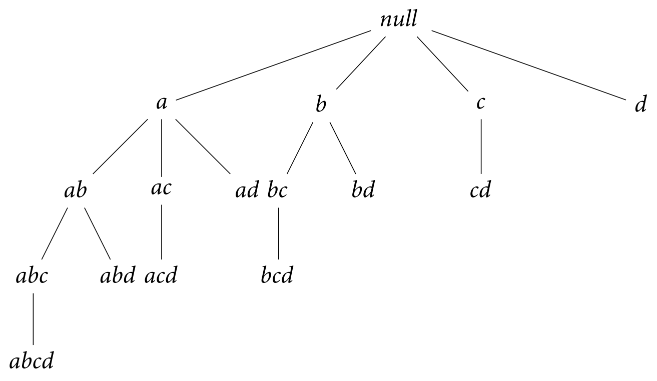

Fig. 1. Part of the enumeration tree.

3. Preliminaries and problem formulation

This section provides some definitions, pruning strategies, and structures used in our algorithms. Given a set with single distinct items $I = \{i_1, i_2, \ldots, i_k\}$, each have its corresponding unit utility $P(i_j)$, here $P(i_j)$ can be larger or less than 0. A quantified transaction T_j is a set of items and their corresponding quantities, $T_j = \langle(i_{j_1}, q_{j_1}), (i_{j_2}, q_{j_2}), \ldots\rangle$, where i_{j_1} represents the first item in transaction T_j, and q_{j_1} is the corresponding quantity of item i_{j_1}. We record a set of quantified transactions $\langle T_1, T_2, \cdots, T_{|D|}\rangle$ as a transaction quantitative database D, where $|D|$ is the number of transactions. If there is an item $i \in I$ satisfying $P(i) < 0$, D is called a database with both positive and negative utilities, denoted as an N-database. Table 1 is a small database with 8 transactions and 7 items. Table 2 shows the unit utility generated by each item. The database shown in Table 1 is obviously an N-database.

To avoid redundant generation, an enumeration tree of the search space is needed, which requires an order of items in *I*. To demonstrate this process, we used alphabetical order. In the beginning, the first layer of the tree is null. Next, we add single items a, b, c, $\cdots$ to the second layer. Next, for each item *i* in the second layer, we combine items *j* with *i* and $j \in \{i \in I | j$ comes after i according to alphabetical order$\}$. For example, b, c, d comes after a. Therefore, a has leaf nodes $\{ab\}$, $\{ac\}$, $\{ad\}$. Then we have $\{ab\}$, $\{ac\}$, $\cdots$, $\{cd\}$ in the third layer. By repeating these operations, we can eventually build a complete enumeration tree. A part of the enumeration tree is shown in Fig. 1. Each child node is called an extension of its parent node. The depth-first and breadth-first traversals can be used to traverse an enumeration tree like that in Fig. 1. In our algorithm in Section 4, we choose the depth-first traversal.

3.1. Periodic pattern

Definition 1 (*Periods*). Given an itemset X, we can get a set of transactions containing X, denoted as $T(X) = \{tid | X \in T_{tid}\} = \{tid_1, tid_2, \cdots, tid_m\}$. We define $PS(X) = \{tid_{i+1} - tid_i | i = 0, 1, 2, \ldots, m\}$, where $tid_0 = 0$, $tid_{m+1} = |D|$. With *PS(X)*, we can get the maximum time interval, the minimum time interval and average time interval of X, which is denoted as *maxPer(X)*, *minPer(X)*, and *avgPer(X)* respectively: *maxPer(X)* = max{*PS(X)*}, *minPer(X)* = min{*PS(X)*}, *avgPer(X)* = $(\sum_{k \in PS(X)} k) \,/\, |PS(X)| = |D| \,/\, (|Supp(X)| + 1)$.

For example, in Table 1, for the itemset $\{c\}$, we have $T(c) = \{1, 2, 3, 6, 7\}$. Then $PS(c) = \{1, 1, 1, 3, 1, 1\}$, and we have $maxPer(\{c\}) = \max\{PS(c)\} = 3$, $minPer(c) = \min\{\mathrm{PS(c)}\} = 1$, $avgPer(c) = |D|/(|Supp(X)| + 1) = 8/(5+1) = 1.333$, $|Supp(X)|$ is the number of transactions containing itemset X.

Definition 2 (*Periodic Pattern*). Given *maxPer*, *minPer*, *minAvg*, and *maxAvg*, which are four thresholds specified by users. For itemset X, if *minPer(X)* ≥ *minPer*, *maxPer(X)* ≤ *maxPer*, and *minAvg* ≤ *avgPer(X)* ≤ *maxAvg*, then X is a periodic pattern, and we say X passes the periodic condition.

Strategy 1 (*maxPer Pruning*). *Given an itemset X, if maxPer(X) > maxPer, then neither X nor its extensions will be a periodic pattern. If maxPer(X) ≤ maxPer, we say that X passes maxPer condition.*

In fact, if X' is an extension of X, $maxPer(X')$ is always larger than $maxPer(X)$, because $T(X') \subseteq T(X)$ is always true, which means $T(X')$ is more sparse.

Strategy 2 (*avgPer Pruning*). *Given an itemset X, if avgPer(X) > maxAvg, then neither X nor its extensions will be a periodic pattern. If $avgPer(X) \leq$ maxAvg, we say X passes avgPer condition.*

The strategy is easy to show. According to the formula $avgPer(X) = |D|/(|Supp(X)| + 1)$ (Fournier-Viger et al., 2016), if $avgPer(X) >$ *maxAvg*, then $|D|/(|Supp(X)| + 1) >$ *maxAvg*, which is equal to $|Supp(X)| < |D|/maxAvg - 1$. And if X' is an extension of X, $|Supp(X')| < |Supp(X)| < |D|/maxAvg{-}1$, then $avgPer(X') >$ *maxAvg*.

3.2. High-utility pattern

Definition 3 (*Utility of an Itemset*). Let an N-database D be given. For an itemset $X = \{i_{x_1}, i_{x_2}, \cdots, i_{x_s}\}$, where i_{x_m} $(m \leq s)$ belongs to $I = \{i_1, i_2, \ldots, i_n\}$, and a transaction T, if $X \in T$, the utility of X in T is $U(X,T) = \sum_{i_{x_m} \subseteq X}(P(i_{x_m}) * q_{x_m})$. The utility of X in the database D is $U(X) = \sum_{(X \subseteq T_j, T_j \subseteq D)} U(X, T_j)$.

Definition 4 (*Positive and Negative Utility of Itemset X*). Let an N-database D be given. Given an itemset X and a transaction T in D, if $X \in T$, we define the positive utility of X in T as $PU(X,T) = \sum_{i_k \subseteq X, P(i_k) > 0}(P(i_k) \ * \ q_k)$. And the negative utility $NU(X,T) = \sum_{i_k \subseteq X, P(i_k) < 0}(P(i_k) \ * \ q_k)$. The positive and negative utilities of X in database D are $PU(X) = \sum_{(X \subseteq T_j, T_j \subseteq D)} PU(X, T_j)$, and $NU(X) = \sum_{(X \subseteq T_j, T_j \subseteq D)} NU(X, T_j)$.

For example, if an itemset $X = \{1, 3\}$, $U(X) = 15 - 3 + 3 - 3 + 3 - 3 = 12$, $PU(X) = 15 + 3 + 3 = 21$, and $NU(X) = -3 - 3 - 3 = -9$. It is obvious that $U(X) = PU(X) + NU(X)$ for any itemset X.

Definition 5 (*High-utility Pattern*). If an itemset X satisfies $U(X) \geq$ *minUtility*, where *minUtility* is a threshold specified by users, then X is called a high-utility pattern. If $U(X) \geq$ *minUtility*, we say itemset X passes utility condition.

In most previous algorithms, there were a lot of pruning strategies in transaction databases without negative utilities. For example, transaction weighed utility (Liu et al., 2005), remaining utility (Liu and Qu, 2012), and estimated utility co-occurrence pruning (EUCP) (Fournier-Viger et al., 2014). However, in a database with both positive and negative utilities, these strategies cannot be used directly. Consequently, we give modified definitions about transaction weighed utility, remaining utility, and EUCP.

Table 3
EUCS.

EUCS				
	a	b	c	d
a		$TWU(\{ab\})$	$TWU(\{ac\})$	$TWU(\{ad\})$
b			$TWU(\{bc\})$	$TWU(\{bd\})$
c				$TWU(\{cd\})$
d				

Definition 6 (*Transaction Weighted Utility TWU in N-database*)**.** Given an itemset X, the transaction weighted utility of X in N-database D is *TWU(X)* $= \sum_{X \in T_j, T_j \subseteq D} PU(T_j)$.

Strategy 3 (*TWU Pruning in N-Database*)**.** *For any single item $i \in I$, if TWU(i) < minUtility, then any itemset X containing i will not be a high-utility itemset.*

Proof. It is simple to demonstrate that for every itemset X containing i, transactions containing X must be a subset of transactions containing i, and *TWU(i) < minUtility*, then *U(X) < TWU(X) < TWU(i) < minUtility*.

In our algorithms, there are two scans of the database. In the first scan of database, we can get *TWU*, *avgPer* and *maxPer* of each single item. Before the second scan, we delete items i such that *TWU(i) < minUtility* or $avgPer(i) >$ *maxAvg* or $maxPer(i) >$ *maxPer* from D and I. The set of remaining items is denoted as I^*. Next, we will give an order to items in I^*, which is shown in Definition 7.

Definition 7 (*Order of Items*)**.** For a given set of items $I^* = \{i_1, i_2, \cdots, i_m\}$, we define an order $\prec$ of items in I^* as follows: for two items i_s, i_t $\in I^*$, firstly, if i_s is a negative item and i_t is a positive item, we define $i_s \prec i_t$; secondly, if both are positive or negative items, if *TWU(i_s) < TWU(i_t)*, we define $i_s \prec i_t$.

Besides the order of single items, we can also define the order of itemsets. Given two itemsets X_1 and X_2. We say that $X_1 \prec X_2$ if $s \prec t$ for all $s \in X_1$, $t \in X_2$.

Definition 8 (*Remaining Utility RU in N-Database*)**.** Let X be an itemset and T_j be a transaction containing X. We define the remaining utility of X in T_j by $RU(X, T_j) = \sum_{i_k \in T_j, i_k \succ X, P(i_k) > 0}(P(i_k) * q_k)$. The remaining utility of X is defined by *RU(X)* $= \sum_{X \in T_j, T_j \in D} RU(X, T_j)$.

Strategy 4 (*RU Pruning in N-Database*)**.***Let X be an itemset. If RU(X) + $U(X)$ < minUtility, then X and its extensions will not be high-utility itemsets. If $RU(X) + U(X) \geq$ minUtility, we say X passes RU condition.*

Proof. It is obvious because we have order when generating itemsets. When we consider extensions of itemset X, only items in set $\{i \in I | i \succ X\}$ will be extended into X, which means utility of any extension of X will be less than $RU(X) + U(X)$.

Strategy 5 (*Estimated Utility Co-occurrence Pruning (EUCP) in N-database*)**.** *For two distinct single items i_{j_1}, i_{j_2}, we store TWU(i_{j_1}, i_{j_2}) in a triangular matrix, and TWU(i_{j_1}, i_{j_2}) can be indexed by (j_1, j_2). The matrix is called estimated utility co-occurrence structure (EUCS). For an itemset X, if it contains items i_{j_1}, i_{j_2} and TWU(i_{j_1}, i_{j_2}) < minUtlity, then X and its extensions will not be high-utility patterns.*

For two itemsets X_1 and X_2 with last items i_{j_1} and i_{j_2}, if *TWU*(i_{j_1}, i_{j_2}) ≥ *minUtility*, we say X_1 and X_2 pass *EUCP*. For example, if we have four items $\{a, b, c, d\}$, *EUCS* is shown in Table 3. In the algorithm PHMN shown in Section 4, we choose the depth-first method to traverse the enumeration tree. Each node of the tree is combined with a PNU-list, which is used to store some related information. The definition is shown below.

Table 4
PNU-list of $\{a, c\}$.

Itemset $\{a, c\}$			
tid	*pu*	*nu*	*ru*
1	15	−3	28
2	3	−3	12
3	3	−3	0

Definition 9 (*PNU-List Lin et al., 2016; Gan et al., 2020c*)**.**The positive-and-negative utility list (PNU-list) of an itemset X is a 2-tuple $(X, list)$, X denotes the itemset name, and *list* denotes the list structure of X. The list structure consists of rows; each row is a 4-tuple, which are the transaction id where X belongs to, positive utility in this transaction, negative utility in this transaction, and remaining utility in this transaction, respectively.

For example, the PNU-list structure of $X = \{ac\}$ in Table 1 is shown in Table 4.

3.3. DU-pruning

Generally, a list of k-itemset X is constructed with two lists of $(k-1)$-itemsets X_1 and X_2. X_1 and X_2 need to have the same prefix, $\tilde{X}$, which ensures that we will not produce the same itemset over and over again. Then X_1, X_2 can be written as $\tilde{X} \cup x$ and $\tilde{X} \cup y$, where $\tilde{X}$ is the common prefix and x, y are single items in the set I. In most algorithms based on the list structure, when we want to get a list of X by combining lists of X_1, X_2, we will verify if $TWU(\{x, y\}) \geq$ *minUtility* firstly. If it is true, then we construct a list of X by combining lists of X_1, X_2. Since *TWU* is a loose upper bound, most itemsets can pass the *TWU* condition, but they are not high-utility itemsets. Although they are not high-utility itemsets, they cannot be pruned, because other itemsets may be produced by X.

For example, assume we have an itemset $X = \{ac\}$, and we know $TWU(a, c) \geq$ *minUtility* and $U(ac) \leq$ *minUtility*. And we have another itemset $X' = \{ab\}$ and $RU(ab) \geq$ *minUtility*, then itemset $\{abc\}$ might be a high-utility itemset. In most algorithms based on list structure, $\{abc\}$ is generated by $\{ab\}$ and $\{ac\}$. Thus, although $X = \{ac\}$ is not a high-utility itemset, a list of $\{ac\}$ still needs to be constructed. Besides, the construction of lists is costly in algorithms based on the utility-list structure, which has been illustrated in Dawar et al. (2017).

It means we construct a list of itemsets that is not our target itemset, and we cannot prune it directly. Because the "useless" construction guarantees the completeness of patterns. In this example, we observe that in the generation process of $\{abc\}$, itemset $\{ab\}$ contributes items a and b, and itemset $\{ac\}$ contributes item c. We already have lists of itemset $\{ab\}$ and $\{c\}$, which contain information about items a, b, c. A new combination idea is shown in Fig. 2. There is a normal part of the enumeration tree on the left side, where we get the list of $\{abc\}$ by combining the lists of $\{ab\}$ and $\{ac\}$. On the right side, we linked node $\{ac\}$ with list of $\{c\}$. Therefore, when we combine nodes $\{ab\}$ and $\{ac\}$ in an enumeration tree, we actually combine lists of $\{ab\}$ and $\{c\}$. In this way, we do not change the enumeration tree, which guarantees the completeness of patterns, but we have fewer constructions.

However, in the process of constructing list of $\{abc\}$, general method (as shown in Fournier-Viger et al., 2016; Liu and Qu, 2012) will combine lists of $\{ab\}$ and $\{ac\}$ firstly, and then delete common prefix $\{a\}$. If we want to get $\{abc\}$ by combining $\{ab\}$ with $\{c\}$, then we need to delete the smaller prefix. $\{a\}$ is prefix of $\{a, b\}$ and $\emptyset$ is prefix of $\{c\}$, and we need to delete $\emptyset$ to get $\{abc\}$. To store smaller prefixes, we propose a modified list structure (Mlist). To know when we need to use the new structure, we propose a new upper bound called the dynamical upper bound (DU). Based on DU and Mlist, we propose a new strategy to prune search space.

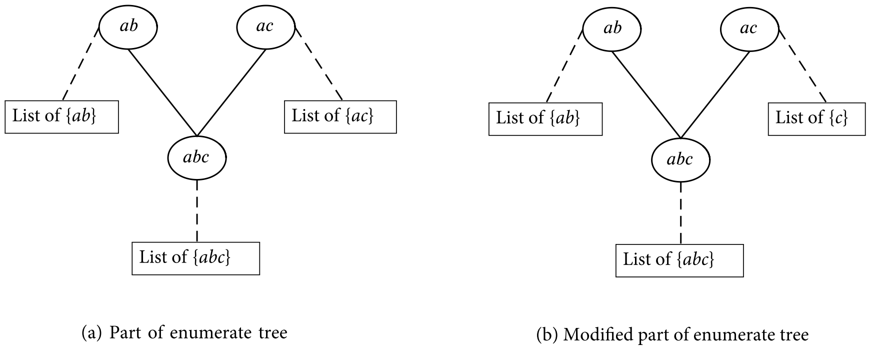

Fig. 2. Parts of the enumeration tree.

Definition 10 (*Mlist*)**.**We define a modified PNU-list (*Mlist*), which is combined with node X in the enumeration tree. *Mlist* is a 4-tuple (X, X', X'-list, prefix of X'), where X is node itemset, which denotes the itemset *Mlist* is combined with in enumeration tree, X' is a subset of X and represents true itemset stored in Mlist. X'*-list* and *prefix-list* are lists of X' and its prefix, where the prefix of an k-itemset is an itemset composed by former $k-1$ items.

In the list of X, $RU(X)$ is the sum of *ru* columns in the list structure of X, which is related to the RU pruning Strategy 4. In Mlist, we will give another value to $RU(Y\text{-}Mlist)$ and $PU(Y\text{-}Mlist)$ to ensure that *Mlist* is identical to the list and the strategies mentioned above can still work. The details of computation are given in the following definition.

Definition 11 (*Dynamical Upper Bound–DU*)**.**Given two nodes X and Y in the same layer of the enumeration tree. X is an itemset going to be extended. $Y \succ X$, X and Y have the same prefix $\tilde{X}$. Then X, Y can be written as $\tilde{X} \cup x$ and $\tilde{X} \cup y$, where x and y are single items in the set I. If Y has a list instead of Mlist, then $DU(Y,X) = RU(Y) + PU(Y) + PU(x)$, else if Y has a Mlist instead of list, then $DU(Y,X) = RU(Y\text{-}Mlist) + PU(Y\text{-}Mlist) + PU(x)$.

For example, in database shown in Table 1, we have two itemsets $A = \{abd\}$ and $B = \{abc\}$. We have $B \prec A$. Because $P(c) < 0$. Then $DU(A,B) = RU(A) + U(A) = 0 + 55 = 55$. As shown in Section 4, when the searching process begins, all single items are combined with lists instead of Mlists, which ensures that we can compute DU value. With Definitions 11 and 10, we propose a new strategy called DU pruning, to reduce the search place.

Lemma 1 (*DU*)**.** *Given two itemsets X and Y in the same layer of the enumeration tree. X is an itemset going to be extended. $Y \succ X$ and X and Y have the same prefix. Let $PU(X \cup Y\text{-}Mlist) = 0$, $RU(X \cup Y\text{-}Mlist) = DU(Y,X)$. Then $RU(X \cup Y) + PU(X \cup Y) \le DU(Y,X)$.*

Proof. If Y is a node combined with a *list*. We have $DU(Y,X) = RU(Y) + PU(Y) + PU(x) = RU(X \cup Y) + PU(Y) + PU(X) \ge RU(X \cup Y) + PU(X \cup Y)$.

If Y is a node combined with a *Mlist*. We will show it by means of mathematical induction. If Y is a 2-itemset, then X is also an 2-itemset because they are in the same layer of the enumeration tree. We write X and Y as $z \cup x$ and $z \cup y$, where x, y, and z are single items. Then we have $DU(Y,X) = RU(Y\text{-}Mlist) + PU(\text{Y-}Mlist) + PU(x) = RU(\text{Y}-Mlist) + PU(x) = RU(z \cup y - Mlist) + PU(x) = DU(y,z) + PU(x)$. Since y, z are single items, they are combined with *lists* not *Mlists*. Therefore, $DU(\text{Y, X}) = DU(\text{y, z}) + PU(x) = RU(y) + PU(\text{y}) + PU(z) + PU(x) = RU(X \cup Y) + PU(y) + PU(z) + PU(x) \ge RU(X \cup Y) + PU(X \cup Y)$.

Assume for $(k-1)$-itemsets X and Y, we have $RU(X \cup Y) + PU(X \cup Y) \le DU(Y, X)$. We show that the inequality holds for k-itemset X and Y. $DU(Y,X) = RU(Y\text{-}Mlist) + PU(Y\text{-}Mlist) + PU(x) = RU(Y\text{-}Mlist) + PU(x)$. Without loss of generality, Y can be considered as a combination of two $(k-1)$-itemsets Y_1 and Y_2. Then we have $DU(Y,X) = RU(Y - Mlist) + PU(x) = RU(Y_1 \cup Y_2\text{-}Mlist) + PU(x) = DU(Y_2, Y_1) + PU(x) \ge RU(Y_1 \cup Y_2) + PU(Y_1 \cup Y_2) + PU(x) = RU(Y) + PU(Y) + PU(x) = RU(X \cup Y) + PU(Y) + PU(x) \ge RU(X \cup Y) + PU(X \cup Y)$.

Strategy 6 (*DU Pruning*)**.** *Given two nodes X and Y in the same layer of enumeration tree. X is an itemset going to be extended. $Y \succ X$ and X and Y have the same prefix. If $DU(Y,X) <$ minUtility, then we do not construct list of $X \cup Y$. If Y is combined with a list and Y has prefix $\tilde{Y}$, then we combine node $X \cup Y$ with Mlist ($X \cup Y$, Y, Y-list, $\tilde{Y}$-list). If Y is combined with a Mlist (Y, Y', Y'-list, $\tilde{Y}'$-list), we combine node $X \cup Y$ with Mlist ($X \cup Y$, Y', Y'-list, $\tilde{Y}'$-list).*

We now illustrate how *DU* pruning strategy works. Firstly, $PU(X \cup Y\text{-}Mlist) = 0$, which makes Mlist identifiable with list. For a given list L, if $PU(L) = 0$, we say L is a Mlist. Because according to item order, items are ordered first according to positive or negative items, and then they are ordered according to *TWU* value. Therefore, negative items will not extend other items behind them; they are used to being extended. Thus, for any k-itemset X, $k \ge 2$, it must contain positive items, which leads to $PU(X) > 0$. If we have a list of k-itemset X, $k \ge 2$, and $PU(X) = 0$, it is a Mlist. Secondly, $RU(X \cup Y\text{-}list) = DU(Y,X)$, which guarantees RU pruning strategy work under Mlist. To show RU pruning strategies still work, we only need to show $RU(X \cup Y\text{-}list) + PU(X \cup Y \text{ - } Mlist) \ge RU(X \cup Y) + PU(X \cup Y)$. And $RU(X \cup Y\text{-}list) + PU(X \cup Y\text{-}Mlist) = DU(Y,X) + 0 = DU(Y,X)$. We already have $DU(Y,X) \ge RU(X \cup Y) + PU(X \cup Y)$ (shown in Lemma 1), that means RU pruning strategy can still work under Mlist.

It is obvious that if $DU(Y,X) <$ *minUtility*, we must have $U(X \cup Y) <$ *minUtility*. It means that the list of $X \cup Y$ is useless. Most importantly, in the computation of $DU(X,Y)$, the information used is stored in the lists of X, Y. It means that we can get the value of $DU(X,Y)$ from lists (or Mlists) of X, Y without having to construct a list of $X \cup Y$. Thus, on the premise of ensuring the completeness of patterns, we can reduce the construct times with the DU pruning strategy.

Assume that we have the following situation: lists of single items $\{a\}$, $\{b\}$, $\{c\}$, $\{abc\}$ is a high-utility itemset but $\{ac\}$ is not, which is possible because utility does not have the download closure property (Chan et al., 2003). In previous algorithms based on list structures like (Fournier-Viger et al., 2016, 2014; Liu and Qu, 2012), to get itemset $\{a,b,c\}$, we need to construct lists of $\{ab\}$, $\{ac\}$, and then combine $\{ab\}$, $\{ac\}$ to get $\{abc\}$. It means to get $\{abc\}$, it is necessary to construct 3 lists $\{ab\}$, $\{ac\}$, and $\{abc\}$. However, with DU-pruning as shown in Strategy 6, before construct lists of $\{ab\}$, $\{ac\}$, we will compute $DU(b,a)$ and $DU(c,a)$, which is tighter upper bound than $TWU(ab)$, $TWU(ac)$, if $DU(c,a) <$ *minUtility*, we do not need to construct the list of $\{ac\}$, instead, we transform the list of $\{c\}$ into the *Mlist* of $\{c\}$. In the process, we only need to construct two lists instead of 3. Fig. 3 depicts the lists required in a mining process with and without the DU pruning strategy.

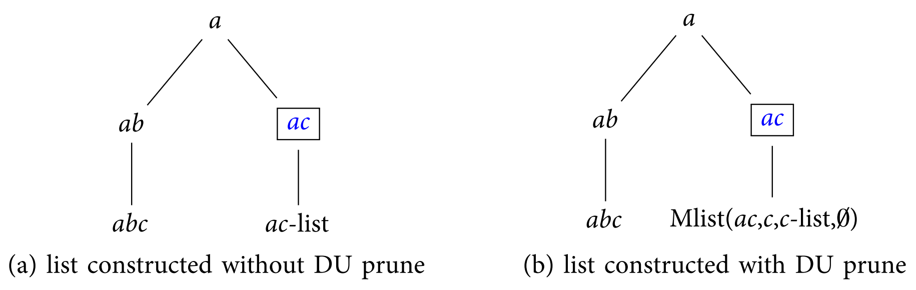
a
ab
ac
abc
ac-list
(a) list constructed without DU prune
a
ab
ac
abc
Mlist(ac,c,c-list,∅)
(b) list constructed with DU prune

Fig. 3. Lists needed to be constructed.

4. Proposed algorithm

Based on Section 3, we show algorithms in this section. We propose the PHMN algorithm to discover periodic high-utility itemsets in a database with both positive and negative utilities. Then we propose an enhanced algorithm called PHMN+, which utilizes the DU pruning strategy.

4.1. Preparation stage

Before formal mining, we need a preparation stage to gather some basic information, which gives a starting point to the search procedure. Algorithm 1 is a preparation procedure for PHMN. In Algorithm 1, it scans the database twice. The first time, it scans the database to calculate *TWU* and periodic information for each item. In the second time, it rearranges the database, by deleting items that satisfy one of the conditions: *TWU(i)* $\leq$ *minUtility*, $maxPer(i) >$ *maxPer*, $avgPer(i) <$ *minAvg*; and reordering items in transactions according to the order $\prec$. In the process of rearranging, it gathers the lists for each single item and *EUCS*. In the end, it calls Algorithm 2 to start formal mining. PHMN+ also gathers the utility of each item, which is the only difference between PHMN and PHMN+ during the preparation stage.

```
Algorithm 1: Preparation procedure
Input: D: N-database; minUtility: thresholds for utility; minPer,
       maxPer, minAvg, maxAvg: periodic thresholds;
Output: a set of periodic high-utility itemsets
1 get $TWU(i)$ value, $maxPer(i)$ and $avgPer(i)$ for each item $i$ by
    scanning D;
2 delete items by TWU, maxPer and avgPer to get $I^*$ and reorder
    the database by $\prec$;
3 get LIST and EUCS;    // LIST is a set of lists of each item in $I^*$
    ;
4 call Search(∅, null, LIST, minPer, maxPer, minAvg, maxAvg)
```

Set *minUtility* $= 30$, *minPer* $= 1$, *maxPer* $= 5$, *minAvg* $= 1$, *maxAvg* $= 3$. After the preparation procedure, the items e, f, g are deleted because $maxPer(e) = 6$, *TWU(f)* $= 6$, *TWU(g)* $= 21$, and database in Table 1 is rearranged. The remaining items have an order of $a \prec b \prec d \prec c$. The rearranged database and estimated utility co-occurrence structure (*EUCS*) are shown in Table 5 and 6, respectively.

4.2. Searching stage

In the searching stage, PHMN has three inputs. *P* is a prefix itemset of length k. *P-list* is a list of itemset *P*. *Lists* are extensions of *P* and their lists. The search algorithm is an iterative function. Each iteration can be divided into two parts. In the first part, verify whether the first itemset X in *Lists* is a periodic high-utility itemset. In the second part, it takes X as new prefix, then generates extensions of X, denoted as *New_list*. Finally, it recalls Algorithm 2 with new parameters: *Search*(X, *X-list*, *New_lists*). PHMN takes all pruning strategies except DU pruning mentioned in this paper. Note that *RU* is computed according to Definition 8, which is different from classical definitions. The pseudocode is shown in Algorithm 2.

Table 5
Rearranged database.

Tid	Item	Quantity
T_1	$a\ b\ d\ c$	5 2 2 1
T_2	$a\ d\ c$	1 1 1
T_3	$a\ c$	1 1
T_4	a	1
T_5	a	1
T_6	$b\ d\ c$	3 3 2
T_7	c	6

Table 6
EUCS.

EUCS				
	a	b	c	d
a		55	73	70
b			109	109
c				124
d				

In PHMN, at the beginning, "∅", items " a, b, d, c" and their "lists" are inputs. $U(a) = 27 < 30$, a is not a periodic high-utility itemset. Next, it verifies if a is an itemset worth to be expended. $RU(a) + U(a) = 79 > 30$, *maxPer(a)* $= 3 < 5, 1 < avg(a) = |D|/(sup(a) + 1) = 4/3 < 3$, then a is chosen to be new prefix going to be extended, then compute *TWU* value of $\{ab,\ ad,\ ac\}$ according to *EUCS* shown in Table 6, then combine list a and lists b, d, c respectively, we get $\{ab,\ ad,\ ac\}$ by 3 list constructions. Then take $\{ab\}$ as prefix, repeat above operation. The enumeration tree generated based on the Algorithm 2–PHMN is shown in Fig. 4(a). Except lists of single items a, b, d, c, PHMN need total 7 list constructions, they are $\{ab\}$, $\{ad\}$, $\{ac\}$, $\{bd\}$, $\{bdc\}$, $\{bc\}$, $\{dc\}$.

The pseudocode of PHMN+ is shown in Algorithm 3. PHMN+ has four inputs, "P", "P-list", "Lists", and "ItemUtility". Here, the last one is the utility of each single item. Compared with PHMN, PHMN+ has a similar main idea in the searching stage. PHMN+ is also an iterative function. And each iteration can be divided into two parts. In the first part, it is the same as with PHMN. In the second part, still taking X as new prefix, it generates extensions of X *New_list*, by using all the pruning strategies mentioned above, containing DU pruning. In the end, it recalls Algorithm 3 with new parameters: *Search*(X, *X-list*, *New_lists*, and *ItemUtility*). The pseudocode is shown in Algorithm 3.

For example, in the beginning of PHMN+, "∅", "∅", "lists" and "utilities" of items a, b, d, c are input. $U(a) = 27 <30$, a is not a periodic high-utility itemset. Similarity, next, verifying if a is an itemset worth to be expended. Since $RU(a) + U(a) = 79 > 30$, $maxPer(a) = 3 < 5, 1 < avg(a) = |D|/(supp(a) + 1) = 4/3 < 3$, $X = \{a\}$ is chosen to be a new prefix going to be extended, compute *TWU* value of $\{ab,\ ad,\ ac\}$ according to *EUCS* structure shown in Table 6. Next, before combining a and $\{b, d, c\}$, it will compute $DU(i, a)$, $i \in \{b, d, c\}$. Since $DU(b, a) > 30$, $DU(c, a) = 0 + 27 = 27 < 30$, then it does not construct the list

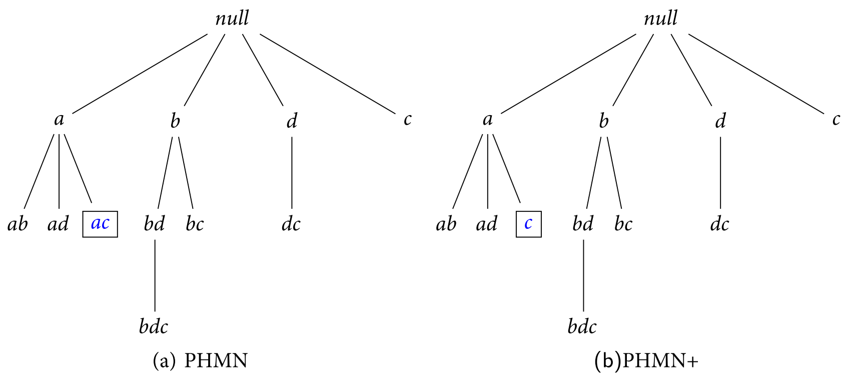

Fig. 4. Enumeration tree based on example database 1.

```
Algorithm 2: Searching procedure of PHMN
Input: P: prefix, P-list: a list of P, Lists: the lists of extensions of
       P, minPer, maxPer, minAvg, maxAvg
Output: a set of periodic high-utility itemsets.
foreach itemset $X$ in Lists do
    if $X$ satisfies the conditions of utility and periodic then
        output $X$
    end
    if $X$ satisfies conditions of RU, maxPer, avgPer then
        New_lists = $\emptyset$;
        New_period = $\emptyset$;
        foreach itemset $Y$ in Lists, and $Y \succ X$ do
            if $X,Y$ pass EUCP then
                $Z = X \cup Y$;
                Z-list = construct($P$, $X$, $Y$);
                New_lists.add(z-list);
            end
        end
        call Search($X$, X-list, New_lists)
    end
end
```

```
Algorithm 3: Searching procedure of PHMN+
Input: P: prefix, P-list: a list of P, Lists: the lists of extensions
       of P, ItemUtility: utility of each single item, minPer,
       maxPer, minAvg, maxAvg.
Output: a set of periodic high-utility itemsets.
foreach itemset $X$ in Lists do
    if $X$ satisfies the conditions of utility and periodic then
        output $X$
    end
    if $X$ satisfies the conditions of RU, maxPer, avgPer then
        New_lists = $\emptyset$;
        New_period = $\emptyset$;
        foreach itemset $Y$ in Lists, and $Y \succ X$ do
            if $X,Y$ pass EUCP then
                compute $DU$(Y,X) ;
                if $DU$(Y,X) pass utility then
                    $Z = X \cup Y$;
                    if Y is combined with a
                      Mlist(Y,$\tilde{Y}$,$\tilde{Y}$.list,$\tilde{Y}$.prefix) then
                        Z-list = construct($\tilde{Y}$.prefix, $X$, $\tilde{Y}$.list);
                    else
                        Z-list = construct($P$, $X$, $Y$)
                    end
                    New_lists.add(Z-list);
                else
                    if Y is combined with a Mlist($Y$, $\tilde{Y}$, $\tilde{Y}$.list,
                      $\tilde{Y}$.prefix) then
                        $X \cup Y$-Mlist = GenMlist1($X$, Y.Mlist,
                          DU(Y,X));
                        New_lists.add($X \cup Y$-Mlist) ;
                    else
                        $X \cup Y$-Mlist = GenMlist2($X$, Y.list, $P$,
                          DU(Y,X));
                        New_lists.add($X \cup Y$-Mlist) ;
                    end
                end
            end
        end
        call Search($X$, X-list, New_lists, ItemUtility)
    end
end
```

of $\{ac\}$, but just combines node $\{ac\}$ with *Mlist* ($\{ac\}$, c, *c-list*, $\emptyset$). The enumeration tree based on Algorithm 3 is shown in Fig. 4(b). Except for the lists of single items a, b, d, c, PHMN+ needs six constructions of lists, that are $\{ab\}$, $\{ad\}$, $\{bd\}$, $\{bdc\}$, $\{bc\}$, and $\{dc\}$. Compared with PHMN, PHMN+ reduces the construction of lists in the mining process.

5. Experiments

In this section, experiments are conducted to evaluate the performance of proposed PHMN algorithm and its enhanced version PHMN+ for mining periodic high utility patterns in datasets with both positive and negative utilities. Besides, the effect of the DU pruning strategy is evaluated. All experiments are performed on a computer with a 7th generation 64-bit Core i5 processor running Windows 10 and 12 GB of free RAM.

5.1. Datasets

Algorithms are evaluated using four datasets, namely accidents_negative, kosarak_negative, pumsb_negative, and connect_negative. These datasets come from different application scenarios and have

Algorithm 4: GenMlist1

```
Input: $X$, $Y.Mlist$, $DU$
Output: $Mlist$ of $X \cup Y$
1 $new\_Mlist = \emptyset$, $new\_Mlist = Y.Mlist$, $new\_Mlist.nodeItemset = X \cup Y$;
2 $RU(new\_Mlist) = DU$, $PU(new\_Mlist) = 0$;
3 return ($new\_Mlist$ )
```

Algorithm 5: GenMlist2

```
Input: $X$, $Y.list$, $P$, $DU$
Output: Mlist of $X \cup Y$
1 $new\_Mlist = \emptyset$, $new\_Mlist.nodeItemset = X \cup Y$,
  $new\_Mlist.trueItemset = Y$, $new\_Mlist.list = Y.list$,
  $new\_Mlist.prefix = P$;
2 $RU(new\_Mlist) = DU$, $PU(new\_Mlist) = 0$;
3 return ($new\_Mlist$)
```

Table 7
Description of parameters.

Parameter description	Symbol
The number of transactions (dataset size)	T
The number of items (maximum items count)	I
Average item count of per transaction (average transaction length)	A
Periodic threshold	P

Table 8
Characteristics of datasets.

Dataset	T	I	A	P
accidents_negative	340,183	468	33.8	1-2000-5-1000
kosarak_negative	990,002	41,270	8.1	1-5000-5-5000
pumsb_negative	49,046	2,113	74	1-3000-1-2000
connect_negative	67,557	129	43	1-3000-1-2000

various characteristics. And they are usually used to evaluate high-utility pattern mining algorithms. Details of these datasets, including the number of transactions, the number of items, and the average item count of per transaction, are given in Table 8. These datasets are real, but with synthetic utility values. For example, log-normal distribution and uniform distribution are used in the generation of external utilities and quantities, which is similar to the setting of previous studies (Ahmed et al., 2009; Fournier-Viger et al., 2014; Lin et al., 2016). More details can be seen in Lin et al. (2016). These datasets can be downloaded at http://www.philippe-fournier-viger.com/spmf.

5.2. Experiments and evaluation on real datasets

The algorithms PHMN and PHMN+ are evaluated in this subsection from the aspects of runtime and memory consumption. The number of patterns to mine in PHMN and PHMN+ is the same, we do not show it in the figure. Periodic thresholds are shown in the last column of Table 8, and threshold 1-2000-5-1000 means that set $minPeriod = 1$, $maxPeriod = 2000$, $minAverage = 5$, and $maxAverage = 1000$. The way of choosing periodic threshold is similar to Fournier-Viger et al. (2016).

Runtime analysis: Fig. 5 shows runtime when varying utility thresholds with given periodic thresholds. It is observed when utility goes down, the runtimes of PHMN and PHMN+ all show a decreasing tendency, and PHMN+ is faster than PHMN in all datasets.It is noted that the performance of algorithms depends on the varied characteristics of datasets. Therefore, increase rates on different datasets are distinct, and specific increase rates are about 10%, 15%, 47% and 50% in four datasets. This indicates the enhanced PHMN+ with DU pruning performs better than PHMN. For example, in Fig. 5(d), runtime of PHMN decreases from 234 s to 60 s, while PHMN+ decreases from 117 s to 26 s, and about half the time is saved by using PHMN+.

Memory analysis: Fig. 6 shows memory when varying utility threshold with given periodic thresholds. It is observed when utility goes down, the result of comparing memory between PHMN and PHMN+ shows a fluctuating state. At most thresholds, PHMN+ uses less memory, and at the remaining thresholds, PHMN uses less memory. However, memory consumption between PHMN and PHMN+ does not have huge differences. Fig. 6(b), memories of PHMN and PHMN+ with five distinct utility thresholds: PHMN uses 1096 MB, 1120 MB, 1120 MB, 1119 MB, and 1121 MB; and PHMN+ uses 1091 MB, 1096 MB, 1097 MB, 1090 MB, and 1134 MB. The differences are negligible, implying that PHMN+ has no additional memory load.

5.3. Experiments and evaluation on synthesis datasets

In order to show the efficiency of PHMN+ and PHMN under different situations. We analyze some synthesis datasets by varying some of the parameters listed in Table 7: T (the number of transactions), I (the number of items), and A (the average item count of per transaction). We vary the number of items from 50, 100, 200, 300, and 500; then we vary the average transaction item count from 5, 10, 20, 40, and 80; and finally, we vary the number of transactions in a dataset from 200k, 400k, 800k, 1000k, and 2000k, to analyze the efficiency in three dimensions: runtime, memory usage, and count of generating lists. To make the results more objective, when varying one parameter, we keep other parameters static. However, when we vary some parameters, the dataset changes at the same time. This means it is not reasonable if we use the same utility threshold (or relative utility threshold value) for different datasets, especially when we vary I or A. Therefore, we use percentages to calculate the analysis runtime and count of generating lists when varying I and A, which is shown in Fig. 7(a) and 8(b). It is helpful to reduce the differences brought about by different utility thresholds.

Analysis when varying maximum items: In Fig. 7, we keep T (the number of transactions), A (the average item count of per transaction), and periodic thresholds unchanged, and adjust the maximum items contained in a dataset from 50, 100, 200, 300, and 500, to observe the performance of PHMN and PHMN+. We choose appropriate but different utility threshold for different dataset, and use formula *improvement percentage* = (PHMN - PHMN+)/PHMN to calculate runtime improvement and list creating improvement, which is shown in Fig. 7(a). In Fig. 7(a), we observe when maximum items is 50, the improvement percentage of runtime and list creating reach a peak (18% and 25%, respectively), and when the maximum are 300 and 500, the improvement percentage is very subtle (a little higher than 0%). Besides, we can observe that the overall trend is that improvement percentages get lower when the number of maximum items increases. Memory usage is shown in Fig. 7(b), and we can see that PHMN+ does not require much memory as a cost to increase efficiency. In most situations, memory usage between PHMN and PHMN+ is similar, and sometimes PHMN+ can save some memory.

Analysis when varying average transaction length: In Fig. 8, we keep T (the number of transactions), I (maximum items count), and periodic thresholds unchanged, and adjust the average transaction length in a dataset from 5, 10, 20, 40, to 80, to observe the performance of PHMN and PHMN+. We choose an appropriate but different utility threshold for different datasets to calculate runtime improvement and list creation improvement, which is shown in Fig. 8(a). In Fig. 8(a), we observe that when the average transaction length is 80, the improvement percentages of runtime and list creation reach a peak (30% and 75%), and when the maximum is 5, the improvement percentage is very subtle (little higher than 0%). Furthermore, we can see that the overall trend is for improvement percentages to increase as average transaction length increases. In Fig. 8(b), memory usage is shown, and we can observe that PHMN+ does not need much memory to get the efficiency

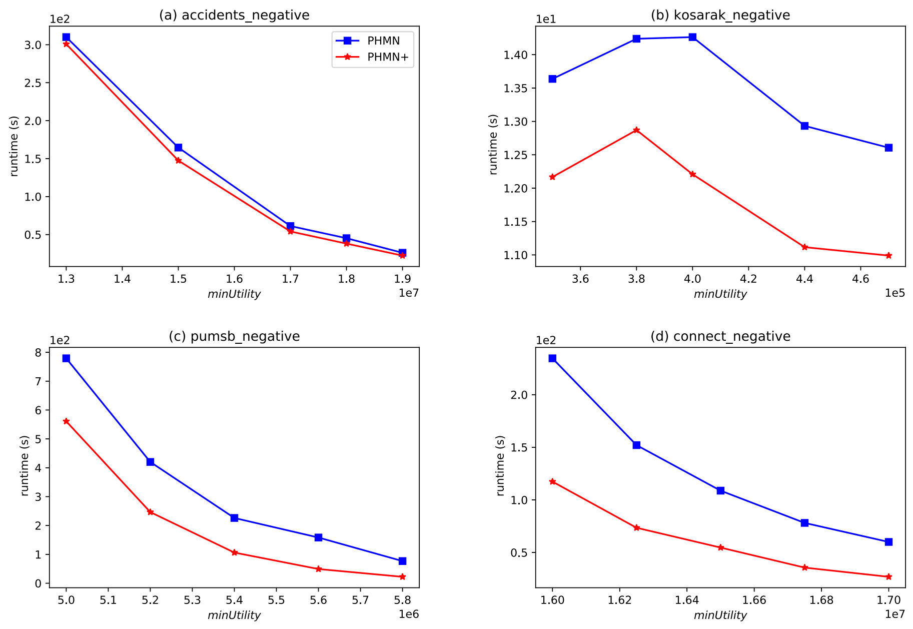

Fig. 5. Runtime when varying *minUtility*. (a) accidents_negative, (b) kosarak_negative, (c) pumsb_negative, (d) connect_negative.

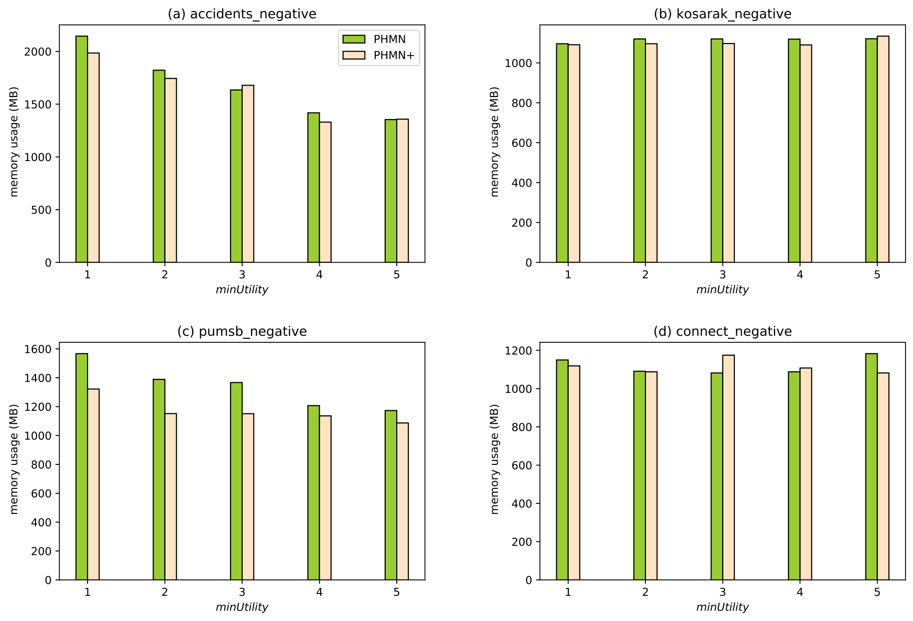

Fig. 6. Memory when varying *minUtility*. (a) accidents_negative, (b) kosarak_negative, (c) pumsb_negative, (d) connect_negative.

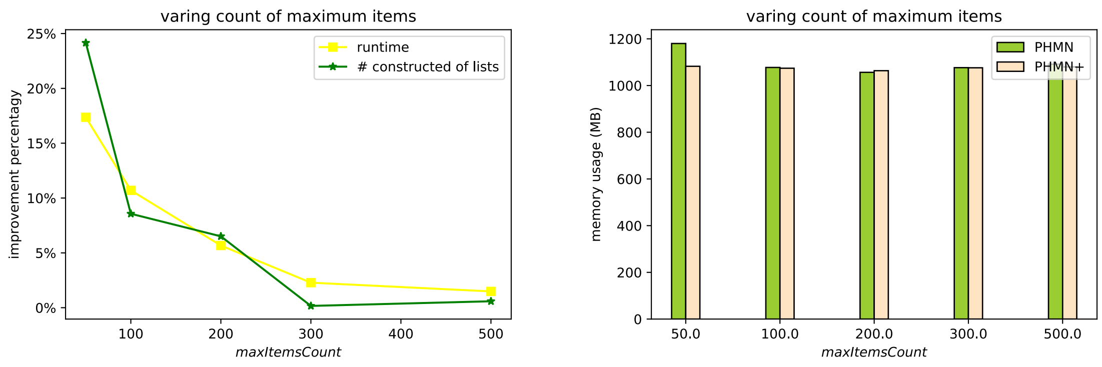

Fig. 7. Results when varying count of maximum items.

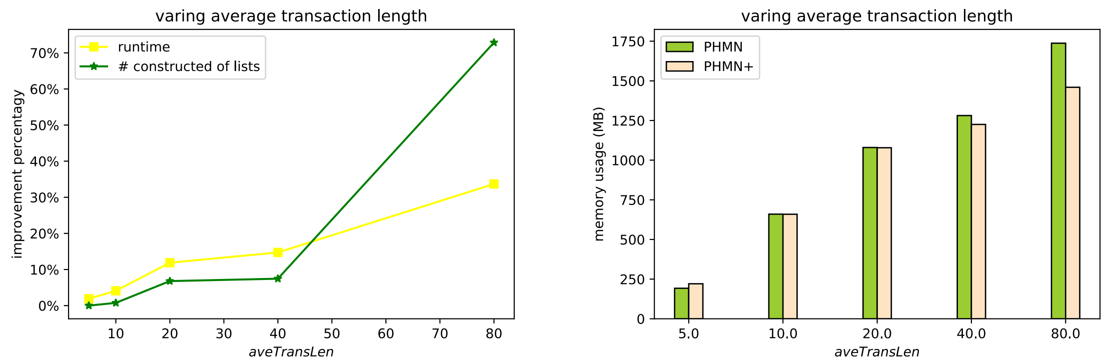

Fig. 8. Results when varying average transaction length.

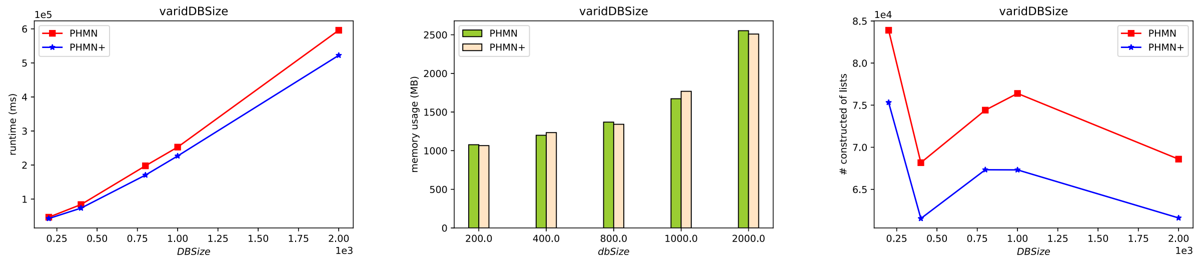

Fig. 9. Results when varying dataset size.

higher. In most situations, memory usage between PHMN and PHMN+ is similar, and sometimes PHMN+ can save some memory.

Analysis when varying dataset size: In Fig. 9, we keep I (maximum items count), A (average item count of per transaction), and periodic thresholds unchanged, and adjust the dataset size in a dataset from 200k, 400k, 800k, 1000k, to 2000k, to observe the performance of PHMN and PHMN+. Because in this situation, I and A are constants, only the dataset size is changing. So, we can use the same relative utility threshold in different datasets, and we choose 0.3 as our relative utility threshold. We can get the *minUtil* which is the relative utility threshold (0.3), multiplied by the total utility of the dataset. Fig. 9(a) shows the runtime of PHMN and PHMN+ under different dataset sizes. We can observe that the runtime goes up when the dataset is bigger. We can see that the bigger the dataset size, the bigger the difference between PHMN and PHMN+, which means that the improvement is more excellent as the dataset size grows and PHMN+ is more scalable than PHMN. Memory usage is depicted in Fig. 9(b), which is similar to the situation depicted in Figs. 7 and 8. The count of list generation is shown in Fig. 9(c).

5.4. *Effect of pruning strategy*

We also evaluate the effectiveness of the DU pruning strategy. Notice that DU pruning works by reducing the number of lists needed to be constructed, so we record the number of lists needed to be constructed and show it in Fig. 10. Combined with Figs. 5 and 10, we can observe that the fewer lists constructed, the more runtime is saved. For example, in Fig. 10(a), about 8.9% lists do not need to be constructed, and the corresponding time improvement shown in Fig. 5(a) is about 10%. Besides, in Fig. 10(d), about 53.3% lists do not need to be constructed, and the corresponding time improvement shown in Fig. 5(d) is about 50%.

Besides, we know PHMN+ can mine periodic high-utility patterns in datasets with both positive and negative utilities, and it is based on list structure. Therefore, PHMN+ gets the ability to deal with dataset without negative utilities. Then if we adjust the period thresholds so that they are too loose to limit patterns. Finally, PHMN+ becomes an algorithm that can only mine high utility patterns in datasets without negative utilities. Then PHMN+ can be compared with the current state-of-the-art FHM algorithm based on list structure. We choose the same four datasets, accidents, kosarak, pumsb, and connect, but without negative utilities (i.e., these datasets are still real datasets but with synthetic utility values. When we generate synthetic utility values, we adjust parameters of distribution so that negative utilities are not generated). These datasets can also be downloaded at http://www.philippe-fournier-viger.com/spmf.

In aspects of runtime, which is shown in Fig. 11 that the DU pruning strategy saves 18.2%, 37.50%, 67.8%, and 42.9%, which shows it is efficient. We know that DU pruning 6 saves time by reducing the number of constructed lists. Therefore, we also compare the numbers of constructing lists between the FHM and PHMN+ in Fig. 13. As expected, the curve of the constructed number is similar to the process runtime; while fewer lists are constructed, more time is saved. Next we compare memory, which is shown in Fig. 12. Memories used in FHM and PHMN+ are similar. This shows the DU pruning strategy does not increase memory load, while it makes runtime faster. In most cases, it even saves some memory.

6. Conclusion

In this paper, we first propose the algorithm PHMN to discover periodic high-utility patterns in databases with both positive and negative utilities. To be more efficient, we propose the DU pruning strategy and an enhanced PHMN+ algorithm based on it. The runtime of PHMN+ is up to two times faster than PHMN, according to experiments. Besides, the DU-pruning strategy is based on list structure, so it can be applied to other algorithms based on list structure. The real world is always more complex than a model, and different domains have distinct goals. Therefore, choosing proper characters for the database will be helpful to get interesting information. In the future, other characters may be combined with PHM in the database, with both positive and negative utilities. Besides, it will be interesting to combine the idea of DU with other techniques, such as tree-based pattern-growth, projected databases, and transaction merging, which may improve the mining efficiency of algorithms.

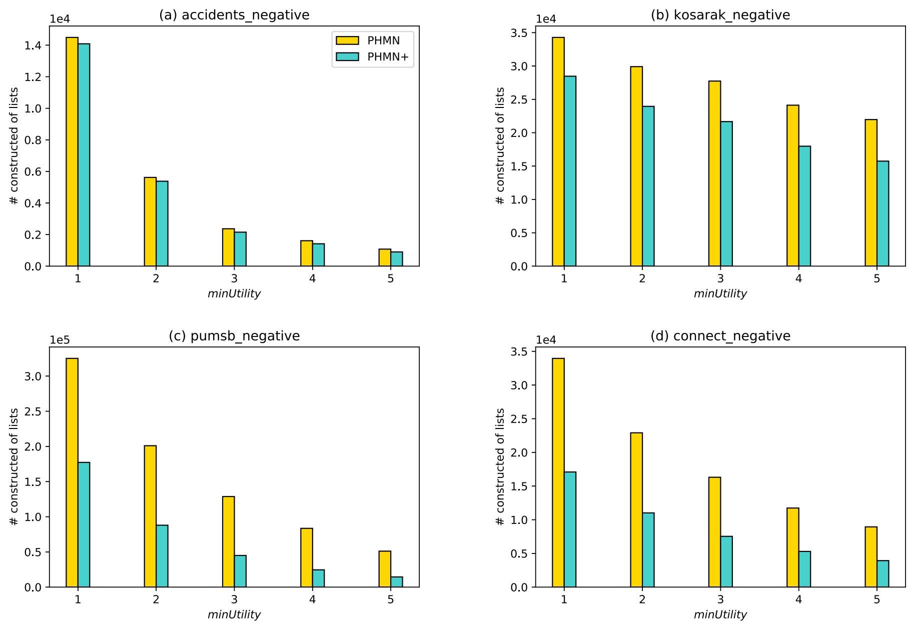

Fig. 10. Number of constructing lists when varying *minUtility*.

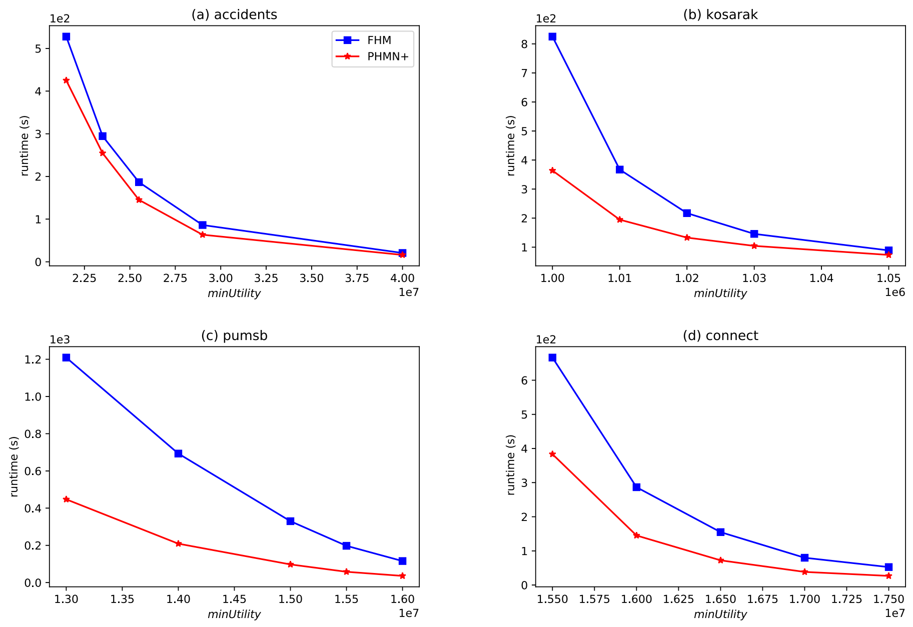

Fig. 11. Runtime when varying *minUtility*. (a) accidents, (b) kosarak, (c) pumsb, (d) connect.

CRediT authorship contribution statement

Fuyin Lai: Conceptualization, Methodology, Writing – original draft. **Xiaojie Zhang:** Data curation, Software. **Guoting Chen:** Supervision, Editing. **Wensheng Gan:** Co-supervision, Writing – review.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Guotng Chen reports financial support was provided by Shenzhen Science and Technology Program (Basic Research Project, No. JCYJ20210324133003011). Wensheng Gan reports financial support was provided by the National Natural Science Foundation of China (Nos. 62002136 and 62272196), Guangzhou Basic and Applied Basic Research Foundation (No. 202102020277), Natural Science Foundation of Guangdong Province (No. 2022A1515011861).

Data availability

Data will be made available on request

Acknowledgments

This work was partially supported by the Shenzhen Science and Technology Program, China (Basic Research Project, No. JCYJ20210324133003011), the National Natural Science Foundation of China (Nos. 62002136 and 62272196), Guangzhou Basic and Applied Basic Research Foundation, China (No. 202102020277), and Natural Science Foundation of Guangdong Province, China (No. 2022A1515011861).

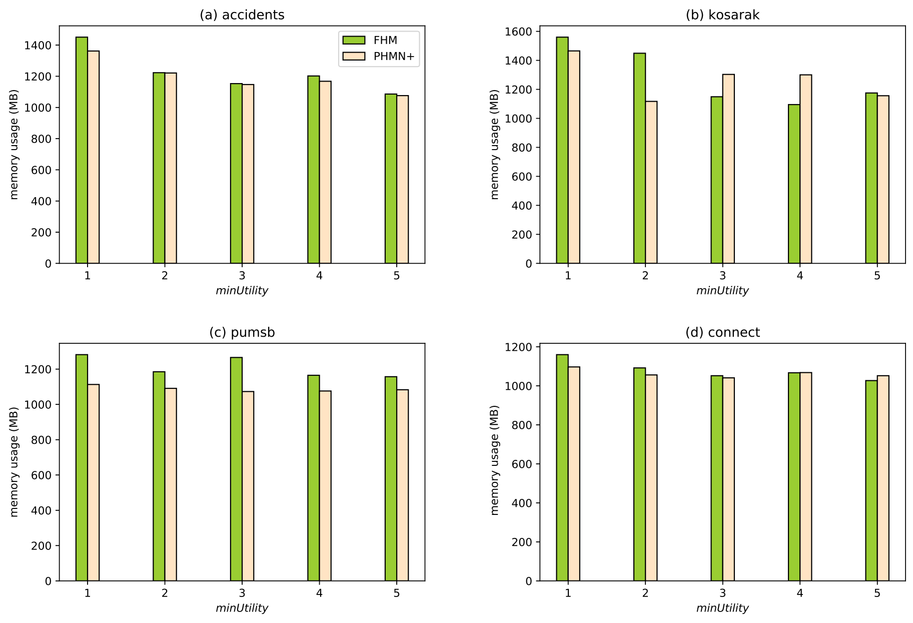

Fig. 12. Momery when varying *minUtility*. (a) accidents, (b) kosarak, (c) pumsb, (d) connect.

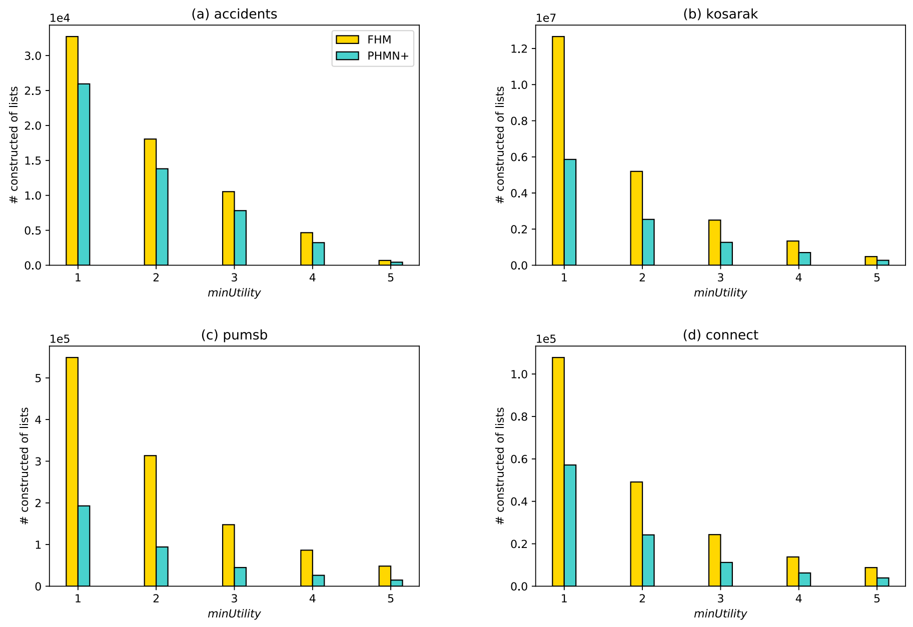

Fig. 13. Number of constructing lists when varying *minUtility*.

References

Agrawal, R., Srikant, R., et al., 1994. Fast algorithms for mining association rules. In: THE 20th International Conference Very Large Data Bases, vol. 1215. Santiago, Chile, pp. 487–499.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Lee, Y.K., 2009. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans. Knowl. Data Eng. 21 (12), 1708–1721.

Amphawan, K., Lenca, P., Surarerks, A., 2009. Mining top-k periodic-frequent pattern from transactional databases without support threshold. In: International Conference on Advances in Information Technology. Springer, pp. 18–29.

Amphawan, K., Surarerks, A., Lenca, P., 2010. Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree. In: Third International Conference on Knowledge Discovery and Data Mining. IEEE, pp. 245–248.

Baek, Y., Yun, U., Kim, H., Kim, J., Vo, B., Truong, T., Deng, Z.H., 2021. Approximate high utility itemset mining in noisy environments. Knowl.-Based Syst. 212, 106596.

Chan, R., Yang, Q., Shen, Y.D., 2003. Mining high utility itemsets. In: Third IEEE International Conference on Data Mining. IEEE Computer Society, p. 19.

Chen, C.M., Chen, L., Gan, W., Qiu, L., Ding, W., 2021. Discovering high utility-occupancy patterns from uncertain data. Inform. Sci. 546, 1208–1229.

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.

Erwin, A., Gopalan, R.P., Achuthan, N., 2008. Efficient mining of high utility itemsets from large datasets. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 554–561.

Fournier-Viger, P., Gan, W., Wu, Y., Nouioua, M., Song, W., Truong, T., Duong, H., 2022. Pattern mining: Current challenges and opportunities. In: International Conference on Database Systems for Advanced Applications. Springer, pp. 34–49.

Fournier-Viger, P., Lin, J.C.W., Duong, Q.H., Dam, T.L., 2016. PHM: Mining periodic high-utility itemsets. In: Industrial Conference on Data Mining. Springer, pp. 64–79.

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: International Symposium on Methodologies for Intelligent Systems. Springer, pp. 83–92.

Gan, W., Lin, J.C.-W., Chao, H.C., Vasilakos, A.V., Yu, P.S., 2020a. Utility-driven data analytics on uncertain data. IEEE Syst. J. 14 (3), 4442–4453.

Gan, W., Lin, J.C.W., Fournier Viger, P., Chao, H.C., Hong, T.P., Fujita, H., 2018. A survey of incremental high-utility itemset mining. Wiley Interdiscipl. Rev.: Data Min. Knowl. Discov. 8 (2), e1242.

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 Trans. Knowl. Data Eng. 33 (04), 1306–1327.

Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Yu, P.S., 2019. A survey of parallel sequential pattern mining. ACM Trans. Knowl. Discov. Data 13 (3), 1–34.

Gan, W., Lin, J.C.-W., Fournier-Viger, P., Chao, H.-C., Yu, P.S., 2020b. HUOPM: High-utility occupancy pattern mining. IEEE Trans. Cybern. 50 (3), 1195–1208.

Gan, W., Lin, J.C.-W., Fournier-Viger, P., Chao, H.-C., Zhan, J., 2017. Mining of frequent patterns with multiple minimum supports. Eng. Appl. Artif. Intell. 60, 83–96.

Gan, W., Wan, S., Chen, J., Chen, C.M., Qiu, L., 2020c. TopHUI: Top-k high-utility itemset mining with negative utility. In: IEEE International Conference on Big Data. IEEE, pp. 5350–5359.

Han, J., Pei, J., Yin, Y., Mao, R., 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. 8 (1), 53–87.

Kiran, R.U., Kitsuregawa, M., 2014. Novel techniques to reduce search space in periodic-frequent pattern mining. In: International Conference on Database Systems for Advanced Applications. Springer, pp. 377–391.

Kiran, R.U., Reddy, P.K., 2009. Mining rare periodic-frequent patterns using multiple minimum supports. In: 15th International Conference on Management of Data. Citeseer, pp. 7–8.

Kiran, R.U., Venkatesh, J., Toyoda, M., Kitsuregawa, M., Reddy, P.K., 2017. Discovering partial periodic-frequent patterns in a transactional database. J. Syst. Softw. 125, 170–182.

Li, H.F., Huang, H.Y., Lee, S.Y., 2011. Fast and memory efficient mining of high-utility itemsets from data streams: With and without negative item profits. Knowl. Inf. Syst. 28 (3), 495–522.

Likhitha, P., Ravikumar, P., Kiran, R.U., Hayamizu, Y., Goda, K., Toyoda, M., Zettsu, K., Shrivastava, S., 2020. Discovering closed periodic-frequent patterns in very large temporal databases. In: IEEE International Conference on Big Data. IEEE, pp. 4700–4709.

Lin, J.C.W., Fournier-Viger, P., Gan, W., 2016. FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits. Knowl.-Based Syst. 111, 283–298.

Lin, J.C.W., Gan, W., Hong, T.P., 2015a. A fast updated algorithm to maintain the discovered high-utility itemsets for transaction modification. Adv. Eng. Inform. 29 (3), 562–574.

Lin, J.C.W., Gan, W., Hong, T.P., Tseng, V.S., 2015b. Efficient algorithms for mining up-to-date high-utility patterns. Adv. Eng. Inform. 29 (3), 648–661.

Liu, Y., Liao, W.k., Choudhary, A., 2005. A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 689–695.

Liu, M., Qu, J., 2012. Mining high utility itemsets without candidate generation. In: Proc. 21st ACM Inter. Conf. Inf. Knowl. Management. pp. 55–64.

Luna, J.M., Fournier-Viger, P., Ventura, S., 2019. Frequent itemset mining: A 25 years review. Wiley Interdiscipl. Rev.: Data Min. Knowl. Discov. 9 (6), e1329.

Nakamura, S., Kiran, R.U., Likhitha, P., Ravikumar, P., Watanobe, Y., Dao, M.S., Zettsu, K., Toyoda, M., 2021. Efficient discovery of partial periodic-frequent patterns in temporal databases. In: International Conference on Database and Expert Systems Applications. Springer, pp. 221–227.

Nguyen, T.D., Nguyen, L.T., Vu, L., Vo, B., Pedrycz, W., 2021. Efficient algorithms for mining closed high utility itemsets in dynamic profit databases. Expert Syst. Appl. 186, 115741.

Nouioua, M., Fournier Viger, P., Wu, C.W., Lin, J.C.W., Gan, W., 2021. FHUQI-miner: Fast high utility quantitative itemset mining. Appl. Intell. 51 (10), 6785–6809.

Rashid, M., Karim, M., Jeong, B.-S., Choi, H.-J., et al., 2012. Efficient mining regularly frequent patterns in transactional databases. In: International Conference on Database Systems for Advanced Applications. Springer, pp. 258–271.

Song, W., Liu, L., Huang, C., 2021. Generalized maximal utility for mining high average-utility itemsets. Knowl. Inf. Syst. 63 (11), 2947–2967.

Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K., 2009. Discovering periodic-frequent patterns in transactional databases. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 242–253.

Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu, P.S., 2012. Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25 (8), 1772–1786.

Tseng, V.S., Wu, C.-W., Shie, B.-E., Yu, P.S., 2010. UP-Growth: An efficient algorithm for high utility itemset mining. In: The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 253–262.

Tung, N., Nguyen, L.T., Nguyen, T.D., Vo, B., 2022. An efficient method for mining multi-level high utility itemsets. Appl. Intell. 52 (5), 5475–5496.

Uday Kiran, R., Krishna Reddy, P., 2010. Towards efficient mining of periodic-frequent patterns in transactional databases. In: International Conference on Database and Expert Systems Applications. Springer, pp. 194–208.

Wang, J.-Z., Chen, Y.-C., Shih, W.-Y., Yang, L., Liu, Y.-S., Huang, J.-L., 2020. Mining high-utility temporal patterns on time interval–based data. ACM Trans. Intell. Syst. Technol. 11 (4), 1–31.

Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12 (3), 372–390.

Zida, S., Fournier Viger, P., Lin, J.C.W., Wu, C., Tseng, V.S., 2017. EFIM: A fast and memory efficient algorithm for high-utility itemset mining. Knowl. Inf. Syst. 51 (2), 595–625.