Expert Systems With Applications 209 (2022) 118214

Contents lists available at ScienceDirect

Expert Systems With Applications

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

# EHMIN: Efficient approach of list based high-utility pattern mining with negative unit profits

Heonho Kim [a,1,2], Taewoong Ryu [a,1,3], Chanhee Lee [a,4], Hyeonmo Kim [a,5], Eunchul Yoon [b,6], Bay Vo [c,7], Jerry Chun-Wei Lin [d,8], Unil Yun [a,*,9]

[a] Department of Computer Engineering, Sejong University, Seoul, Republic of Korea
[b] Department of Electronics Engineering, Konkuk University, Seoul, Republic of Korea
[c] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Viet Nam
[d] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

ARTICLE INFO

*Keywords:*
Data mining
High-utility pattern mining
Negative unit profit
List-based data structure

ABSTRACT

High-utility pattern mining is an important sub-literature in the data mining literature. This literature discusses the discovery of useful pattern information from large databases by considering not only supports of patterns but also profits and quantities of items. This literature has the potential to be applied to various problems in the real world, so many methods for the improvement of the algorithm performance have been studied. Moreover, there have also been attempts to extend the flexibility of this literature. The traditional approaches in this literature considered the positive unit profits of items in a given database only. However, this literature can take extended flexibility into account by considering negative as well as positive unit profits of the items. In this paper, we suggest an efficient approach for mining high-utility patterns with negative unit profits. Moreover, the experimental performance tests, which are performed on various real and synthetic datasets in this paper, show that the proposed algorithm has a better performance than the state-of-the-art methods in this literature in terms of the runtime, memory usage, and scalability.

## 1. Introduction

Many people use information techniques today that have been developed and widely propagated widely. The information techniques were mainly used by the public in order to search useful information. However, as the use of them has been extended, the size of the data that are generated by the public users have become larger and larger. The data mining literature, which analyzes the large accumulated data and extracts the useful information, has recently received a lot of interest for this reason. In the literature, there is the sub-literature, which involves pattern mining, that extracts the useful pattern information by analyzing the combinations of items in a given database. For the extraction of pattern information, the features of the patterns that can be expressed numerically are used as the criteria of the extraction. For this reason, the literature can be classified into various methods depending on the criteria. The most general feature is the frequency, which is equal to the support, and frequent pattern mining is a method using it. The algorithms of the method count the supports of the patterns in a given database and find patterns that have larger support than a user-defined threshold, which is the minimum support. The method has been used

* Corresponding author.
*E-mail address:* yunei@sejong.ac.kr (U. Yun).
[1] First author.
[2] ORCID: 0000-0002-7666-9857.
[3] ORCID: 0000-0002-1927-2896.
[4] ORCID: 0000-0001-5933-7896.
[5] ORCID: 0000-0002-9682-0040.
[6] ORCID: 0000-0002-6321-516X.
[7] ORCID: 0000-0002-4018-660X.
[8] ORCID: 0000-0001-8768-9709.
[9] ORCID: 0000-0002-3720-0861.

https://doi.org/10.1016/j.eswa.2022.118214
Received 15 July 2021; Received in revised form 29 June 2022; Accepted 16 July 2022
Available online 30 July 2022

continuously in order to analyze various data, such as DNA sequence data (Deng, Chen, Li, & Xiong, 2019), travel recommendation data (Kumar & Thangamani, 2019), indoor trajectory data (Chen, Yuan, Qiu, & Pi, 2019), network data (Cafaro, Epicoco, & Pulimeno, 2019), and the behavior data of an organization (Chapela-Campa, Mucientes, & Lama, 2019). However, the method has a limitation, because all the items have the same weight, which include importance, unit profit, and cost. Weighted frequent pattern mining (Zhang, Zhang, Yin, Yuan, Wu, & Luo, 2019), which is a method that considers a different weight for each item, emerged as an extended method. It extracts frequent and profitable pattern information, but it also has a limitation, which is the quantity of each item in a transaction cannot be considered.

High-utility pattern mining (Baek et al., 2021; Gan, Lin, Fournier-Viger, Chao, Tseng, & Yu, 2021; Gan, Lin, Fournier-Viger, Chao, & Yu, 2019b; Lin, Yang, Fournier-Viger, & Hong, 2019), which is a method that considers not only weights of the items but also the quantities of the items in each transaction, emerged as a more extended method. There are many other methods that are extended from frequent pattern mining, such as top-k pattern mining (Pyun & Yun, 2014; Zhang, Yang, Wu, Cheng, Xie, & Lin, 2019; Zhu, Liu, Li, Zhang, & Wen, 2018), uncertain pattern mining (Ahmed, Ahmed, Samiullah, Adnan, & Leung, 2016; Baek, Yun, Yoon & Fournier-Viger, 2019), representative pattern mining (Lee & Yun, 2018a; Tianrui, Mingqi, & Bin, 2018), average-utility pattern mining (Kim, Yun, Baek, Kim, Vo, Yoon, & Fujita, 2021), sequential pattern mining (Gan, Lin, Fournier-Viger, Chao, & Yu, 2019a), periodic pattern mining (Kim, Yun, Vo, Lin, & Pedrycz, 2020; Kim, Yun, Yoon, Lin, & Fournier-Viger, 2020), and erasable pattern mining (Lee & Yun, 2018b; Yun et al., 2019a; Nam, Yun, Yoon, & Lin, 2020). However, high-utility pattern mining is one of the methods that keep being developed. Moreover, there is also an attempt to extend the flexibility of the method. The traditional approaches of the method consider the positive unit profits of the items only in a given database. However, there are many cases in the real world where the negative unit profits for items should be considered. For example, assume that discount events are held for a certain period of time for a chainstore. There are discount events, such as *Open*, *Summer Season*, and *Black Friday*. Also, there are discount amounts, such as 50 % and $5 less. The product management systems in real-world business domains, such as a retail store or discount outlet, can then manage the profits of products with discount as negative (i.e. *Open Sale 5$ less* and *Black Friday* −50 %). In this case, many items may have negative unit profits when the net profit information of the items should be analyzed as opposed to the selling price information. In the real-life business field, various promotional events are held to promote customers. Regarding this situation, if an item with negative utility is extracted as a pattern, it is meaningful information that is different from the existing high utility patterns. This is because the utility of the pattern exceeded the threshold, even though some items reduced the utility of the pattern. For example, in the sales data recorded during the promotion of dairy products, bread and milk cause negative profits because they are discounted. On the other hand, eggs are considered positive profits because they are not discounted. This information is important in retail if bread and eggs (or products not eligible for dairy promotions) are sold together and can generate a lot of revenue. Promotional discounts of chainstore are another example. The utility of coffee becomes negative due to the promotional event. However, coffee is extracted as a pattern with cakes, cookies, or other beverages, which increases total profit. Therefore, the extracted patterns can be helpful in decision making to continue the event or plan the next sale. In other words, real-life firms can set up or modify sales strategies for product bundles that are profitable when sold together. In addition, vendors may have concerns about additional costs caused by negative utility when conducting promotional events to attract customers or promote products. Pattern mining, which considers negative utility, can help maximize profitability when conducting future promotional events by analyzing sales data. Meanwhile, data needs to be processed quickly in order to effectively analyze the rapidly changing purchasing trends of consumers. Therefore, a fast and efficient negative utility pattern mining algorithm is required in order to quickly analyze a database that contains negative profit and maximize the business profit.

High-utility pattern mining can be applied to various cases by considering both the positive and negative unit profits as opposed to the positive unit profits only. The problem of considering negative unit profits is important in high utility itemset mining. The existing algorithms, which do not consider the negative unit profits, extract the wrong pattern results from databases with negative unit profits because their pruning techniques are established only when there are no negative unit profits in given databases. The importance of the problem is discussed in the latest journal articles about high utility itemset mining with negative unit profits (Chu, Tseng, & Liang, 2009; Krishnamoorthy, 2018; Lin, Fournier-Viger, & Gan, 2016). In this paper, we propose an algorithm for the efficient mining of high-utility patterns that consider negative unit profits, which is called EHMIN (Efficient High-utility pattern MIning with Negative unit profits). The main contributions are as follows. 1) Propose a more efficient mechanism in order to construct conditional data structures as opposed to the existing algorithms. We explain the entire process with various examples and detailed pseudo codes in order to help clearly understand them. 2) Perform experimental evaluations with both real and synthetic datasets. The analysis about the results shows that the suggested method has a better performance than the state-of-the-art approaches.

This paper is structured as follows. The studies that are related to our approach are explained in Section 2. The definitions of high-utility pattern mining with negative unit profits are introduced in Section 3, and the construction and the mining process of our approach are described in detail with various examples. Moreover, the proposed algorithm is explained via its pseudo code. In Section 4, the proposed algorithm is compared with the state-of-the-art methods in terms of the runtime, memory usage, and scalability. Finally, the conclusion and future directions are represented in Section 5.

## 2. Related work

In this section, we introduce the studies that are related to the approach that we propose. The representative algorithms and the research flow about high-utility pattern mining are described, and high-utility pattern mining with negative unit profits is then introduced. We clarify the differences between the proposed method and the existing methods through this.

### 2.1. High-utility pattern mining

Frequent pattern mining is one of the most fundamental methods in the pattern mining literature. The purpose of the method is to extract useful pattern information using supports of patterns in a given database. However, there are limitations with this method, because all the items have the same importance, and the quantities of the items in each transaction are not considered. Thus, high-utility pattern mining, which is a method that considers that the items are assigned real number weights and that each item in transactions has its quantity, emerged in order to solve the limitations.

The two-phase algorithm (Liu, Liao, & Choudhary, 2005) is one of the earliest algorithms about high-utility pattern mining. It suggested the concept of Transaction Weighted Utility (TWU), which is used to calculate the over-estimated utilities for its efficient mining process. In addition, it operates in 2 phases. In the first phase, it extracts the high-TWU patterns by using the anti-monotone property that the TWUs of the patterns have. Subsequently, in the second phase, it extracts high-utility patterns from high-TWU patterns by calculating their actual utilities. It is also one of the level-wise candidate generation approaches, and other approaches include UMining & UMining_H (Yao & Hamilton, 2006), FUM & DCG+ (Li, Yeh, & Chang, 2008), and GPA (Lan, Hong, & Tseng, 2012).

The level-wise candidate generation approaches suffer from scalability issues, so the tree-based algorithms are proposed for this reason, which include IHUP (Ahmed, Tanbeer, Jeong, & Lee, 2009), HUC-Prune (Ahmed, Tanbeer, Jeong, & Lee, 2011), UP-Growth (Tseng, Wu, Shie, & Yu, 2010), and UP-Growth+ (Tseng et al., 2012). IHUP is experimentally proven to have better mining performance than the two-phase algorithm and the level-wise candidate generation approaches. Subsequently, list-based algorithms are suggested, such as HUI-Miner (Liu & Qu, 2012), FHM (Fournier-Viger, Wu, Zida, & Tseng, 2014), and HUP-Miner (Krishnamoorthy, 2015). HUI-Miner is one of the earliest algorithms that employed the list-based data structure, and it is experimentally proven to have a better performance than the tree-based algorithms. FHM and HUP-Miner are the extended algorithms from the HUI-Miner algorithm by adding pruning strategies. However, more extended approaches have been suggested, such as PIHUP (Lee, Yun, Lee, & Yoon, 2018), MPM (Yun, Kim, Yoon, & Fujita, 2018), and IIHUM (Yun et al., 2019b).

### 2.2. High-utility pattern mining with negative unit profits

In traditional algorithms in high-utility pattern mining, items only have positive unit profits. However, there are items in the real world with not only positive but also negative unit profits. In addition, the upper bound of the existing high utility pattern mining approaches does not satisfy the anti-monotone property in the database with negative utility. If the mining is conducted without upper bounds, it is inefficient, because all the patterns that can be created should be checked. Therefore, many studies about negative utility pattern mining have been conducted for new upper bound and efficient mining algorithms. HUINIV-Mine (Chu et al., 2009) is proposed in order to mine high-utility patterns with negative unit profits. It is an extended algorithm from the two-phase algorithm and the level-wise candidate generation approaches. It is a revised TWU anti-monotone technique that can be used in order to reduce its search space. However, it has a poor performance on large datasets and on low thresholds, because it adopted the level-wise candidate generation. Subsequently, tree-based algorithms were proposed, such as ENIN (Singh, Shakya, Singh, & Biswas, 2018), UP-GNIV (Subramanian & Premalatha, 2015), and EHNL (Singh, Kumar, Singh, Shakya, & Biswas, 2019), because the level-wise candidate generation approaches suffer from scalability issues. EHNL adopted length constraints for certain purposes that are required in the real world. Moreover, MHUI-BIT & MHUI-TID (Li, Huang, & Lee, 2011) and TS-HOUN (Lan, Hong, Huang, & Tseng, 2014) emerged in order to consider stream data and temporal data. Meanwhile, methods that considered negative item values have been suggested in sequential pattern mining, such as Topk-NSP (Dong, Qiu, Lu, Cao, & Xu, 2019), F-NSP+ (Dong, Gong, & Cao, 2018), HUNSPM (Xu, Li, & Dong, 2018), and HUSP-NIV (Xu, Dong, Xu, & Dong, 2017).

FHN (Lin et al., 2016) is suggested by extending the basic idea of HUI-Miner. It is a data structure that seperately stores the positive and negative item utilities, which is the PNU-list, and it then adopted the existing pruning strategies, such as U-Prune (Krishnamoorthy, 2015; Liu & Qu, 2012), EUCS-Prune (Fournier-Viger et al., 2014), and LA-Prune (Krishnamoorthy, 2015). In addition, it is experimentally proven to have a better performance than HUINIV-Mine. Finally, the GHUM (Krishnamoorthy, 2018) emerged by extending the FHN algorithm. It presented its additional pruning strategies, which include PTWU-Prune and *N*-Prune. In this paper, the state-of-the-art algorithms, which include the FHN and GHUM, are considered as comparisons, and an efficient approach in order to integrate the exiting pruning strategies and to improve the mining performance is proposed.

## 3. Mining High-utility patterns with negative unit profits

We introduce everything that is needed in order to perform our approach in this section. Firstly, the preliminaries for mining high-utility patterns with negative unit profits are described using Tables 1 and 2. Next, the procedure of the proposed approach is described in order to help understand the details. Subsequently, the construction and the mining process are explained in detail with practical examples. Finally, the pseudo codes, which illustrate our approach, are shown in order to explain how the approach is implemented.

**Table 1**
Example Database and the Utility Results.

| TID | Transaction | Quantity (IU) | Utility | PTU |
|---|---|---|---|---|
| 1 | A, B, D, H | 2, 3, 1, 1 | 4, 3, 3, -1 | 10 |
| 2 | A, C, E, H | 2, 4, 2, 3 | 4, 4, -2, -3 | 8 |
| 3 | B, C, D, E, F | 6, 3, 1, 3, 2 | 6, 3, 3, -3, 10 | 22 |
| 4 | A, B, C, G | 4, 3, 3, 2 | 8, 3, 3, -2 | 14 |
| 5 | B, D, E, G, H | 4, 4, 1, 2, 1 | 4, 12, -1, -2, -1 | 16 |

**Table 2**
Item Profits.

| Item | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Profit (EU) | 2 | 1 | 1 | 3 | -1 | 5 | -1 | -1 |

### 3.1. Preliminaries

When $\boldsymbol{DB}$ is a given database and $\boldsymbol{T_k}$ is a transaction with TID $k$, $\boldsymbol{DB}$ is expressed as follows: $\boldsymbol{DB} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \cdots, \boldsymbol{T}_m\}$. Each transaction in $\boldsymbol{DB}$ is a set of items. Let a set of all distinct items in $\boldsymbol{DB}$ be $\boldsymbol{I} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_3, \cdots, \boldsymbol{i}_n\}$. A transaction, which is denoted as $\boldsymbol{T_k}$, is then a subset of $\boldsymbol{I}$ ($\boldsymbol{T_k} \subseteq \boldsymbol{I}$). A pattern $\boldsymbol{X}$ is assumed to describe the general case of the definitions. Pattern $\boldsymbol{X}$ is an undetermined variable pattern, which can be any subset of itemset $\boldsymbol{I}$. A pattern $\boldsymbol{X}$ is also a subset of $\boldsymbol{I}$ ($\boldsymbol{X} \subseteq \boldsymbol{I}$), and $\boldsymbol{X} \subseteq \boldsymbol{T_k}$, which means that $\boldsymbol{X}$ appears in $\boldsymbol{T_k}$. In addition, the support of pattern $\boldsymbol{X}$ indicates the number of transactions where pattern $\boldsymbol{X}$ appears. Table 1 shows an example of $\boldsymbol{DB}$. The basic definitions for mining high-utility patterns with negative unit profits refer to previous studies (Fournier-Viger, Wu, Zida, & Tseng, 2014, Lin et al., 2016, and Krishnamoorthy, 2018).

**Definition 1.** (*(External Utility)*) Each item in **DB**, $\mathbf{i_x} \in \mathbf{I}$, is assigned its weight, which is called the external utility. The external utility of an item, $\mathbf{i_x}$, is denoted as $\mathbf{EU}(\mathbf{i_x})$. Table 2 shows an example of the external utilities for items in **DB**. For example, $\mathbf{EU}(\mathbf{D}) = 3$ in Table 2.

If the items indicate the products for sale, the external utilities of the items indicate the profit of the products. The external utilities are all positive unit profits in the traditional approaches, so the external utilities with negative unit profits also exist, which are shown in Table 2. Let a set of items whose external utilities are greater than 0 be $\boldsymbol{PI}$ and a set of items whose external utilities are$<0$ be $\boldsymbol{NI}$, so $\boldsymbol{PI} \cap \boldsymbol{NI} = \varnothing$ and $\boldsymbol{PI} \cup \boldsymbol{NI} = \boldsymbol{I}$. For example, $\boldsymbol{PI} = \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{F}\}$ and $\boldsymbol{NI} = \{\boldsymbol{E}, \boldsymbol{G}, \boldsymbol{H}\}$, which are shown in Table 2.

**Definition 2.** (*(Internal Utility)*) All the items that emerged in the transactions have their quantities, which are called the internal utility. $\mathbf{IU}(\mathbf{i_x}, \mathbf{T_k})$ indicates the internal utility of an item, $\mathbf{i_x}$, in a transaction, $\mathbf{T_k}$. For example, $\mathbf{IU}(\mathbf{D}, \mathbf{T}_5) = 4$, which is shown in Table 1.

**Definition 3.** (*(Utility of an Item in a Transaction).*) $\mathbf{U}(\mathbf{i_x}, \mathbf{T_k})$ represents the utility of an item, $\mathbf{i_x}$, in a transaction, $\mathbf{T_k}$, and it is calculated by multiplying the external utility and internal utility of the item.

$$U(i_x, T_k) = EU(i_x) \times IU(i_x, T_k) \tag{1}$$

For example, $\boldsymbol{U}(\boldsymbol{D}, \boldsymbol{T}_5) = \boldsymbol{EU}(\boldsymbol{D}) \times \boldsymbol{IU}(\boldsymbol{D}, \boldsymbol{T}_5) = 3 \times 4 = 12$, which is illustrated in Table 1.

**Definition 4.** (*(Utility of a Pattern in a Transaction)*) The utility of a pattern, **X**, which emerged in a transaction, $\mathbf{T_k}$, is denoted as $\mathbf{U}(\mathbf{X}, \mathbf{T_k})$,

and it is calculated as the sum of the utilities of all the items that made up the pattern.

$$U(X,T_k)=\sum_{i_x\in X}U(i_x,T_k) \tag{2}$$

For example, $\boldsymbol{U}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_5)=\boldsymbol{U}(\boldsymbol{D},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{E},\boldsymbol{T}_5)=12+(-1)=11$, which is shown in Table 1.

**Definition 5.** (*(Positive Utility of Pattern in a Transaction)*) The positive utility of a pattern, **X**, which emerged in a transaction, $\mathbf{T_k}$, is denoted as $\mathbf{PU}(\mathbf{X},\mathbf{T_k})$, and it is calculated as the sum of the utilities of all the items which are included in **PI** and made up the pattern.

$$PU(X,T_k)=\sum_{i_x\in X\wedge i_x\in PI}U(i_x,T_k) \tag{3}$$

For example, $\boldsymbol{PU}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_5)=\boldsymbol{U}(\boldsymbol{D},\boldsymbol{T}_5)=12$, which is presented in Table 1.

**Definition 6.** (*(Utility of Pattern)*) The utility of a pattern, **X**, in **DB** is denoted as **U**(**X**), and it is calculated as the sum of the utilities of the pattern in all transactions where the pattern emerged.

$$U(X)=\sum_{X\subseteq T_k\in DB}U(X,T_k) \tag{4}$$

For example, $\boldsymbol{U}(\{\boldsymbol{D},\boldsymbol{E}\})=\boldsymbol{U}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_3)+\boldsymbol{U}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_5)=0+11=11$, which is depicted in Table 1.

**Definition 7.** (*(Positive Utility of Pattern)*) The positive utility of a pattern, **X**, in **DB** is denoted as **PU**(**X**), and it is calculated as the sum of the positive utilities of the pattern in all the transactions where the pattern emerged.

$$PU(X)=\sum_{X\subseteq T_k\in DB}PU(X,T_k) \tag{5}$$

For example, $\boldsymbol{PU}(\{\boldsymbol{D},\boldsymbol{E}\})=\boldsymbol{PU}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_3)+\boldsymbol{PU}(\{\boldsymbol{D},\boldsymbol{E}\},\boldsymbol{T}_5)=3+12=15$, which is presented in Table 1.

**Definition 8.** (*(Transaction Utility)*) The utility of a transaction, $\mathbf{T_k}$, is denoted as $\mathbf{TU}(\mathbf{T_k})$ and calculated as the sum of the utilities of the items belonging to $\mathbf{T_k}$.

$$TU(T_k)=\sum_{i_x\in T_k}U(i_x,T_k) \tag{6}$$

For example,$\boldsymbol{TU}(\boldsymbol{T}_5)=\boldsymbol{U}(\boldsymbol{B},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{D},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{E},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{G},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{H},\boldsymbol{T}_5)=4+12+(-1)+(-2)+(-1)=12$ in Table 1.

**Definition 9.** (*(Positive Transaction Utility)*) The positive utility of a transaction, $\mathbf{T_k}$, is denoted as $\mathbf{PTU}(\mathbf{T_k})$, and it is calculated as the sum of the positive utilities of the items belonging to $\mathbf{T_k}$.

$$PTU(T_k)=\sum_{i_x\in T_k\wedge i_x\in PI}U(i_x,T_k) \tag{7}$$

For example, $\boldsymbol{PTU}(\boldsymbol{T}_5)=\boldsymbol{U}(\boldsymbol{B},\boldsymbol{T}_5)+\boldsymbol{U}(\boldsymbol{D},\boldsymbol{T}_5)=4+12=16$ in Table 1.

**Definition 10.** (*(Positive Transaction Weighted Utility)*) The positive transaction-weighted utility of a pattern, **X**, in **DB** is denoted as **PTWU**(**X**), and it is calculated as the sum of the positive utilities of all transactions where the pattern emerged.

$$PTWU(X)=\sum_{X\subseteq T_k\in DB}PTU(T_k) \tag{8}$$

For example, $\boldsymbol{PTWU}(\{\boldsymbol{D},\boldsymbol{E}\})=\boldsymbol{PTU}(\boldsymbol{T}_3)+\boldsymbol{PTU}(\boldsymbol{T}_5)=22+16=38$ in Table 1.

**Definition 11.** (*(Processing Order of Items)*) The processing order represents the sorting order of the list in the algorithm. The $\succ$ symbol indicates the sorting order of the items in processing orders, and all the distinct items, which are the elements of **I**, are sorted according to the following rules. (i) $\mathbf{PI}\succ\mathbf{NI}$, (ii) all the elements of **PI** are sorted according to a PTWU-ascending order, and (iii) all the elements of **NI** are sorted according to a support-ascending order. In the proposed algorithm in this paper, **I** is always processed according to this order. This ordering was presented earlier in the FHN (Lin, Fournier-Viger, & Gan, 2016). For example, Table 3 shows the PTWUs and the supports of all items in Table 1. The ordering of the items in Table 1 is then: $\mathbf{F}\succ\mathbf{A}\succ\mathbf{C}\succ\mathbf{D}\succ\mathbf{B}\succ\mathbf{G}\succ\mathbf{E}\succ\mathbf{H}$.

**Table 3**
PTWUs and Supports of the Items.

| Item | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| **PTWU** | 32 | 62 | 44 | 48 | 46 | 22 | 30 | 34 |
| **Support** | 1 | 4 | 3 | 3 | 3 | 1 | 2 | 3 |

| | F | A | C | D | B | G | E |
|---|---|---|---|---|---|---|---|
| A | 0 | | | | | | |
| C | 22 | 22 | | | | | |
| D | 22 | 10 | 22 | | | | |
| B | 22 | 24 | 36 | 48 | | | |
| G | 0 | 14 | 14 | 16 | 30 | | |
| E | 22 | 8 | 30 | 38 | 38 | 16 | |
| H | 0 | 18 | 8 | 26 | 26 | 16 | 24 |

**Fig. 1.** Estimated Utility Co-occurrence Structure (EUCS).

**Definition 12.** (*(Minimum Utility)*) The user-defined percentage threshold is denoted as $\delta$, and the minimum utility is denoted as **minUtil**. **minUtil** is calculated as the product of the sum of all the positive transaction utilities in **DB** and $\delta$.

$$minUtil=\delta\times\sum_{T_k\in DB}PTU(T_k) \tag{9}$$

For example, where $\delta=20\%$, $\boldsymbol{minUtil}=0.2\times(10+8+22+14+16)=0.2\times70=14$, which is shown in Table 1.

**Definition 13.** (*(High-utility Pattern)*) A pattern, **X**, is called as a high-utility pattern if $\mathbf{U}(\mathbf{X})\geq\mathbf{minUtil}$. If $\mathbf{U}(\mathbf{X})\langle\mathbf{minUtil}$, **X** is then a low-utility pattern. For example, where $\mathbf{minUtil}=14$, $\mathbf{U}(\{\mathbf{D},\mathbf{E}\})=11<14$, so {**D**, **E**} is a low-utility pattern. In addition, $\mathbf{U}(\{\mathbf{D},\mathbf{B}\})=31\geq14$, so {**D**, **B**} is a high-utility pattern.

**Definition 14.** (*(Estimated Utility Co-occurrence Structure)*) A triangular matrix that holds the PTWUs for all pairs of the items is called the Estimated Utility Co-occurrence Structure (EUCS).

$$EUCS[i_x,i_y]=PTWU(\{i_x,i_y\})\,(i_x,i_y\in I,i_x\succ i_y) \tag{10}$$

Fig. 1 shows the EUCS for the database in Table 1. In Fig. 1, the column designates $\boldsymbol{i_x}$ and the row indicates $\boldsymbol{i_y}$. Each entry at $(\boldsymbol{i_x},\boldsymbol{i_y})$ holds $\boldsymbol{EUCS}[\boldsymbol{i_x},\boldsymbol{i_y}]$. EUCS was presented in FHM (Fournier-Viger et al, 2014) for the efficiency of traditional high-utility pattern mining, and it was then adopted in GHUM (Krishnamoorthy, 2018) and FHN (Lin et al., 2016) for mining high-utility patterns with negative unit profits.

**Definition 15.** (*(Remaining Pattern)*) Assume that there is a pattern, **X**, emerged in a transaction $\mathbf{T_k}$. A set of items after **X** in the transaction sorted by a processing order, $\mathbf{T_k'}$, is called the remaining pattern of **X** in

**Table 4**
High-utility Patterns from the Database in Table 1 where..***minUtil*** = 14

| High-utility Pattern | Utility | High-utility Pattern | Utility | High-utility Pattern | Utility | High-utility Pattern | Utility |
|---|---|---|---|---|---|---|---|
| F, C, D | 16 | F, D, B | 19 | A, C, B | 14 | D, B, G | 14 |
| F, C, D, B | 22 | F, D, B, E | 16 | A, B | 18 | D, B, E | 21 |
| F, C, D, B, E | 19 | F, B | 16 | C, B | 15 | D, B, E, H | 14 |
| F, C, B | 19 | A | 16 | D | 18 | D, B, H | 20 |
| F, C, B, E | 16 | A, C | 19 | D, B | 31 | B | 16 |

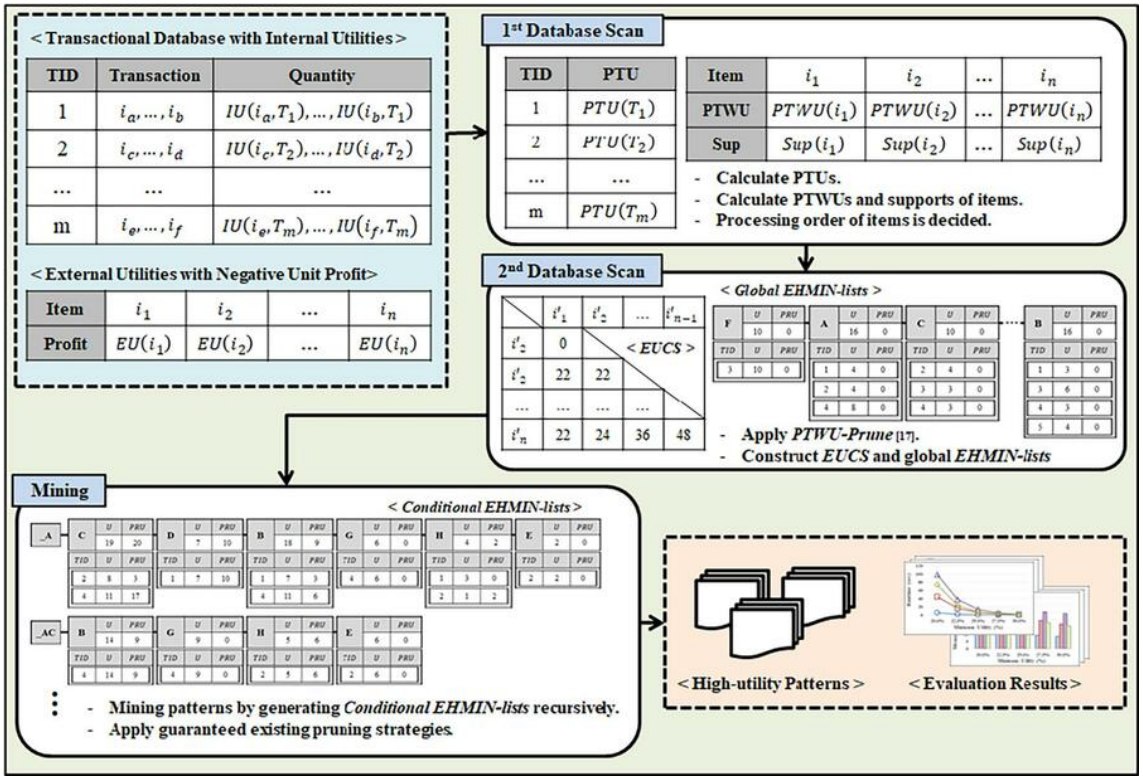


**Fig. 2.** The Overall Process of the Proposed Approach.

$\mathbf{T_k}$, which is denoted as $\mathbf{T_k}\backslash\mathbf{X}$. For example, $\mathbf{T}_3\backslash\{\mathbf{F},\mathbf{D}\} = \{\mathbf{B},\mathbf{E}\}$, which is shown in Table 1, because the sorted $\mathbf{T}_3$ is $\{\mathbf{F},\mathbf{C},\mathbf{D},\mathbf{B},\mathbf{E}\}$.

**Definition 16.** (*(Positive Remaining Utility of Pattern in a Transaction)*) The positive remaining utility of a pattern, $\mathbf{X}$, in a transaction, $\mathbf{T_k}$, is denoted as $\mathbf{PRU}(\mathbf{X},\mathbf{T_k})$, and it is calculated as the sum of the utilities of all the items that are included in $\mathbf{T_k}\backslash\mathbf{X}$ and **PI**.

$$PRU(X,T_k) = \sum_{i_x \in (T_k \backslash X) \wedge i_x \in PI} U(i_x,T_k) \tag{11}$$

For example, $\boldsymbol{PRU}(\{\boldsymbol{C},\boldsymbol{D}\},\boldsymbol{T}_3) = \boldsymbol{U}(\boldsymbol{B},\boldsymbol{T}_3) = 6$, which is shown in Table 1.

**Definition 17.** (*(Positive Remaining Utility)*) The positive remaining utility of $\mathbf{X}$ in **DB** is denoted as $\mathbf{PRU}(\mathbf{X})$, and it is calculated as the sum of all $\mathbf{PRU}(\mathbf{X},\mathbf{T_k})$ where $\mathbf{X}$ emerged.

$$PRU(X) = \sum_{X \subseteq T_k \in DB} PRU(X,T_k) \tag{12}$$

For example, $\boldsymbol{PRU}(\{\boldsymbol{A},\boldsymbol{C}\}) = \boldsymbol{PRU}(\{\boldsymbol{A},\boldsymbol{C}\},\boldsymbol{T}_2) + \boldsymbol{PRU}(\{\boldsymbol{A},\boldsymbol{C}\},\boldsymbol{T}_4) = 0+3 = 3$ in Table 1.

Table 4 represents 20 patterns that are mined as high-utility patterns from the database in Table 1 where ***minUtil*** = 14. The high-utility patterns can be extracted from various data and thresholds that consider negative unit profits via the definitions that are introduced in this section.

### 3.2. Overall process of mining High-utility patterns with negative unit profit

In this part, we describe the overall process of the proposed approach, which is the EHMIN. It requires a transactional database with internal and external utility information including negative unit profits. Also, it needs a user-defined threshold as its input data, and it finally outputs high-utility patterns and the evaluation results.

Fig. 2 indicates the entire process for EHMIN. It can be divided into 3 steps, which include *1st Database Scan*, *2nd Database Scan*, and *Mining*. In first step, which is *1st Database Scan*, the input data is scanned, and PTUs for each transaction for PTWUs and supports for the items are calculated. ***minUtil*** is then obtained from the sum of all the PTUs, and the processing order for all the items is decided. After that, the index numbers are assigned to all the items by the processing order in this step, the items will be processed by their index numbers. Next, in the second step, which is *2nd Database Scan*, the given database is scanned one more time. According to the scan, the global EHMIN-lists are sorted by the processing order, and EUCS are constructed. The global EHMIN-lists are then perused in order to calculate the PRUs for them. In addition, unpromising items are pruned by a pruning strategy, which is PTWU-Prune, that will be explained soon from the global EHMIN-lists. At the end of this step, the algorithm is ready to mine the valid patterns.

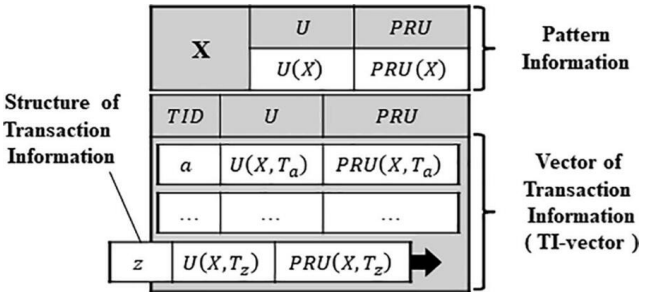


**Fig. 3.** The Structure of an EHMIN-list.

Finally, in the third step, which is *Mining*, high-utility patterns in the current EHMIN-lists are recorded as the mining results, and the conditional EHMIN-lists are then constructed by combining each pair of current EHMIN-lists. In the construction process, the unpromising patterns are pruned by the remaining pruning strategies. After that, the process of this step is recursively repeated. By performing these 3 steps, all high-utility patterns and the performance results of the EHMIN are obtained.

### 3.3. Construction process of data structures with vectors of transaction information

We introduce in this section a data structure for mining high-utility patterns with negative unit profits and explain the detailed process of *1st Database Scan* and *2nd Database Scan* of the EHMIN with practical examples. In addition, any concepts and techniques that are used in the process are also introduced.

**Definition 18.** (*(EHMIN-list)*) An EHMIN-list has the item name or the index number, the utility (U), the positive remaining utility (PRU), and the vector of transaction information, which is a TI-vector. A TI-vector is a vector container for $<$ TID, U, PRU $>$ structures. The main difference this data structure has is that it holds the transaction information of an item as an element of a vector container for more efficient operations. Fig. 3 shows the structure of an EHMIN-list. Given a completely constructed EHMIN-list, the utility of its pattern and the over-estimated utility of its super pattern are available. Thus, each EHMIN-list tells whether its pattern is a high-utility pattern or a low-utility pattern as well as whether its pattern is a promising pattern or an unpromising pattern. The EHMIN-lists are classified by the construction process and their use, which include global EHMIN-lists and conditional EHMIN-lists. The construction processes are introduced in section 3.3 and section 3.4. The global EHMIN-lists are needed in order to hold 1-itemsets information, and they are constructed directly from a given database by two scans. In the first scan, the item information, such as supports, positive transaction weighted utilities (PTWUs), the number of items, the transaction information, such as transaction utilities (TUs), the positive transaction utilities (PTUs), the sum of the TUs, and the minimum utility (**minUtil)** are obtained. By using the item and the transaction information, the processing order for the scanned database is decided. Subsequently, the global EHMIN-lists are constructed in the processing order by scanning the database one more time. At the same time, the construction of the EUCS is also performed. Once the construction of the global EHMIN-lists is all done, the valid 1-itemsets are ready to be extracted. Meanwhile, the conditional EHMIN-lists are needed to hold the **k**-itemsets ($\mathbf{k} \geq 2$) information, and they are constructed from the previously constructed EHMIN-lists.

**Example 1.** Here is a practical example for the construction of global EHMIN-lists. Fig. 4 sequentially shows the construction process of the global EHMIN-lists by scanning the database in Table 1. Fig. 4(a) represents after the first transaction in the database, $\mathbf{T}_1$, is inserted. $\mathbf{T}_1$, which consists of **A, B, D**, and **H**, is sorted in the order of $\mathbf{A} \succ \mathbf{D} \succ \mathbf{B} \succ \mathbf{H}$. This is because {**A**}, {**B**}, and {**D**}, which have a positive utility, are sorted in the ascending order of PTWU, and H, which has a negative utility, is then placed. Next, $\mathbf{T}_1$ is sorted according to the reversed order of the processing order, and the EHMIN-lists for the items are then constructed and sequentially inserted into a set of global EHMIN-lists. The reason they are inserted in the reversed order is that the positive remaining utilities (PRUs) should be simultaneously calculated. For instance, the utility of {**H**} in $\mathbf{T}_1$ is $-1$. The the utility is negative, so the

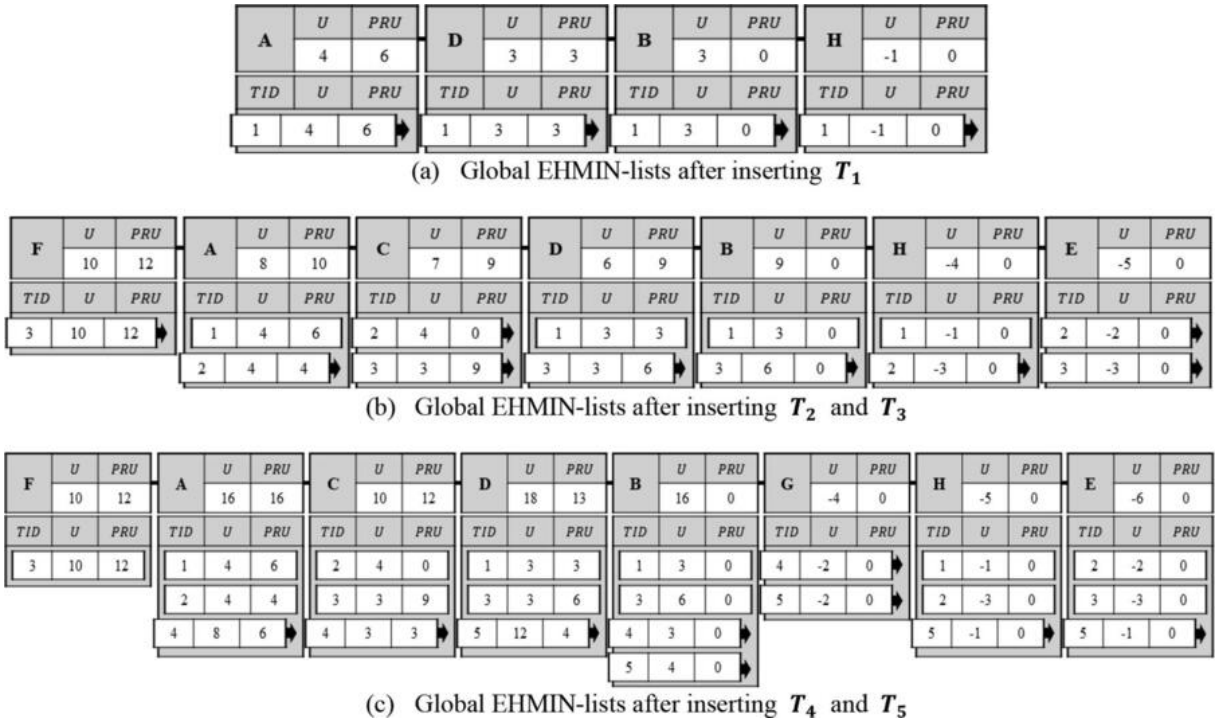


**Fig. 4.** The Construction Process of the Global EHMIN-lists.

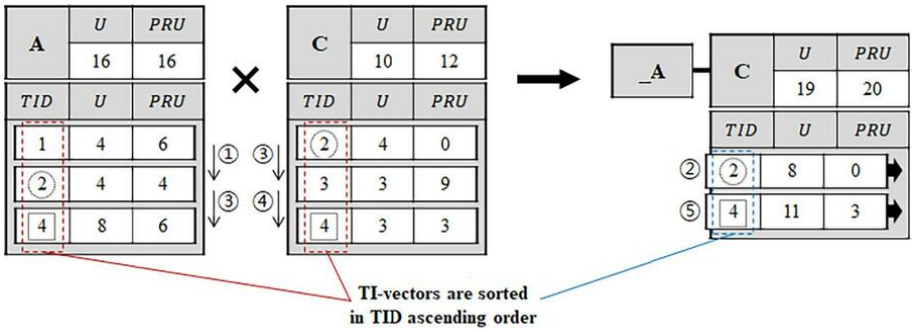


**Fig. 5.** A Search Process on Two TI-vectors.

positive remaining utility is zero, and $< 1, -1, 0 >$ is inserted into the list of {**H**}. The item {**B**} is then processed. In $\mathbf{T}_1$, the utility of {**B**} is 3, and there is no positive utility that appears after {**B**}. Therefore, $<1, 3, 0 >$ is inserted into the list of {**B**}. Next, item {**D**} is processed. In $\mathbf{T}_1$, the utility of {**D**} is 3, and {**B**} with positive utility appears after {**D**}. Therefore, PRU of {**D**} becomes 3, the utility of {**B**}, and $< 1, 3, 3 >$ is created in the list of {**D**}. Finally, {**A**}, which is the first item in sorted $\mathbf{T}_1$, is processed. The utility of {**A**} is 4, and the items, which have a positive utility and appear after {**A**}, are {**B**} and {**D**}. Therefore, PRU of {**A**} becomes 6, which is the sum utility of {**B**} and {**D**}, and $< 1, 4, 6 >$ is inserted into the list of {**A**}. Fig. 4(b) illustrates the lists after two additional transactions in the database, $\mathbf{T}_2$ and $\mathbf{T}_3$, are inserted. The inserted transactions are also sorted by the reversed order of the processing order, and the EHMIN-lists are constructed and inserted into the set of global EHMIN-lists sequentially. Fig. 4(c) shows the lists after all transactions in the database are inserted using the same method. It is the final form of the global EHMIN-lists construction, and the high-utility 1-itemsets are ready to be mined.

*PTWU-Prune* is a pruning technique that is proposed in the GHUM (Krishnamoorthy, 2018). A pattern, $X$, is a low-utility pattern, and all the super patterns of $X$ are also low-utility patterns if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minutil}$. According to Eqs. (4) and (5), $\boldsymbol{U}(X) \leq \boldsymbol{PU}(X)$ is always valid. $X \subseteq T_k$ means that $X$ emerged in $T_k$, so $\boldsymbol{PU}(X, T_k) \leq \boldsymbol{PTU}(T_k)$ according to Eqs. (3) and (7). $\boldsymbol{PU}(X) \leq \boldsymbol{PTWU}(X)$ is then valid according to Eqs (5) and (8), and $\boldsymbol{U}(X) \leq \boldsymbol{PU}(X) \leq \boldsymbol{PTWU}(X)$ is valid. Thus, if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minUtil}$ is true, $\boldsymbol{U}(X)\langle\boldsymbol{minUtil}$ is also true. Therefore, $X$ is a low-utility pattern if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minUtil}$. Let any super pattern of $X$ be $X'$. Then, there are only 2 cases in $T_k(X \subseteq T_k)$: $X \subseteq X' \subseteq T_k$ or $X' \cap T_k = \varnothing$, and $\boldsymbol{PTWU}(X') \leq \boldsymbol{PTWU}(X)$ is valid. Thus, if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minUtil}$ is true, $\boldsymbol{U}(X') \leq \boldsymbol{PTWU}(X')\langle\boldsymbol{minUtil}$ is also true. Therefore, $X$ is an unpromising pattern if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minUtil}$. Once the database scans are all completed, all global EHMIN-lists are completely constructed. However, the lists for mining high-utility patterns in all global EHMIN-lists are not all helpful. Patterns that have the lower positive transaction weighted utility (PTWU) than ***minutil*** are all low-utility patterns, and they are all unpromising patterns. Therefore, the *PTWU-Prune* should be performed before the next process, which is the pattern growth and mining, begins.

### 3.4. Mining process on data structures with vectors of transaction information

We explain the third step of EHMIN, which is the *Mining*, in this section with practical figures and examples. In addition, any concepts and adopted pruning techniques that are used in the process are also introduced. *A-Prune* is a pruning property that is proposed in the GHUM (Krishnamoorthy, 2018). Assume a pattern, $X$, and any super pattern of the pattern, $X'$. If $X' - X \subseteq \boldsymbol{NI}$ and $X' - X \neq \varnothing$ are valid, $\boldsymbol{U}(X')\langle\boldsymbol{U}(X)$ is also valid. This technique is assuming the processing order in Definition 11. Assume that $X' - X = Y \subseteq \boldsymbol{NI}$, so $\boldsymbol{U}(X') = \boldsymbol{U}(X) + \boldsymbol{U}(Y)$ according to Eqs. (2) and (4), and $\boldsymbol{U}(Y)\langle 0$ because $Y \subseteq \boldsymbol{NI}$ and $Y \neq \varnothing$. Therefore, $\boldsymbol{U}(X')\langle\boldsymbol{U}(X)$ is valid when $X' - X \subseteq \boldsymbol{NI}$ and $X' - X \neq \varnothing$. Next, *EUCS-Prune* is a pruning technique that is proposed in the FHM (Fournier-Viger, Wu, Zida, & Tseng, 2014). It uses EUCS that is defined in Definition 14. A 2-itemset, $X = \{i_x, i_y\}$, is a low-utility and an unpromising pattern if $\boldsymbol{EUCS}[i_x, i_y]\langle\boldsymbol{minutil}$. This technique is proposed in FHN (Lin, Fournier-Viger, & Gan, 2016), and it is proofed as follows. According to Eq. (10), $\boldsymbol{EUCS}[i_x, i_y] = \boldsymbol{PTWU}(X = \{i_x, i_y\})$, and *PTWU-Prune*, if $\boldsymbol{PTWU}(X)\langle\boldsymbol{minutil}$, $X$ is a low-utility pattern. Therefore, this technique is true. Next, *U-Prune* is a pruning technique that is proposed in HUI-Miner (Liu & Qu, 2012), and it has been adopted in many approaches for high-utility pattern mining. Let a pattern be $X$. All the super patterns of $X$ are low-utility patterns if $\boldsymbol{U}(X) + \boldsymbol{PRU}(X)\langle\boldsymbol{minutil}$. This technique was adopted in the GHUM (Krishnamoorthy, 2018) and the FHN (Lin, Fournier-Viger, & Gan, 2016). Let any super pattern of $X$ be $X'$. If $X' \cap \boldsymbol{NI} = \varnothing$, $\boldsymbol{U}(X') \geq \boldsymbol{U}(X)$ is valid according to Eqs. (2) and (4). Then, $\boldsymbol{U}(X' - X) = \boldsymbol{U}(X') - \boldsymbol{U}(X) \leq \boldsymbol{PRU}(X)$ because $X' - X \subseteq T_k \backslash X$ in $T_k(X' \subseteq T_k)$. Thus, $\boldsymbol{U}(X') \leq \boldsymbol{U}(X) + \boldsymbol{PRU}(X)$ is always valid when $X' \cap \boldsymbol{NI} = \varnothing$. Meanwhile, if $X' \cap \boldsymbol{NI} \neq \varnothing$, $X' - X \subseteq \boldsymbol{NI}$ because the pattern growth is processed in the processing order. Then, $\boldsymbol{PRU}(X) = 0$, which is according to Eqs. (11) and (12), and $\boldsymbol{U}(X')\langle\boldsymbol{U}(X)$ (*A-Prune*). Thus, $\boldsymbol{U}(X')\langle\boldsymbol{U}(X) + \boldsymbol{PRU}(X)$ is always valid when $X' \cap \boldsymbol{NI} \neq \varnothing$, so $\boldsymbol{U}(X') \leq \boldsymbol{U}(X) + \boldsymbol{PRU}(X)$ is always valid. Therefore, if $\boldsymbol{U}(X) + \boldsymbol{PRU}(X)\langle\boldsymbol{minutil}$, $\boldsymbol{U}(X')\langle\boldsymbol{minutil}$.

Conditional EHMIN-lists are needed in order to hold the $k$-itemsets ($k \geq 2$) information, which are constructed from previously constructed the EHMIN-lists. The construction of a conditional EHMIN-list means combining two patterns. Assume that a set of EHMIN-lists for $(k-1)$-itemsets is already constructed. In this set, each pair of lists are selected and sequentially combined. Whether the patterns can be combined depends on whether the common transaction-ids (TIDs) where the patterns emerged exist. So, when a pair of lists is selected, the TI-vectors for the two patterns are compared, and the common transactions are found. By calculating information of the common transactions, the conditional EHMIN-list is constructed. The total construction time of the conditional EHMIN-lists occupies a great deal of the total mining time, so any reduction of the construction time can be a good way to reduce the total runtime. An approach for the reduction of the construction time is to prune the conditional EHMIN-lists of the unpromising patterns. However, in high-utility pattern mining, the utilities of the patterns don't satisfy the anti-monotone technique, because the utility of a pattern can be larger than the utility of the sub pattern. Thus, there are pruning strategies, such as *U-Prune* and *EUCS-Prune*. Meanwhile, another approach for the reduction of the construction time is to adopt a fast comparing method. The pruning strategies reduce the number of constructions, because they adopt an efficient comparing method, which reduces the runtime for a construction.

**Lemma 1.** All the constructed EHMIN-lists already have their TI-

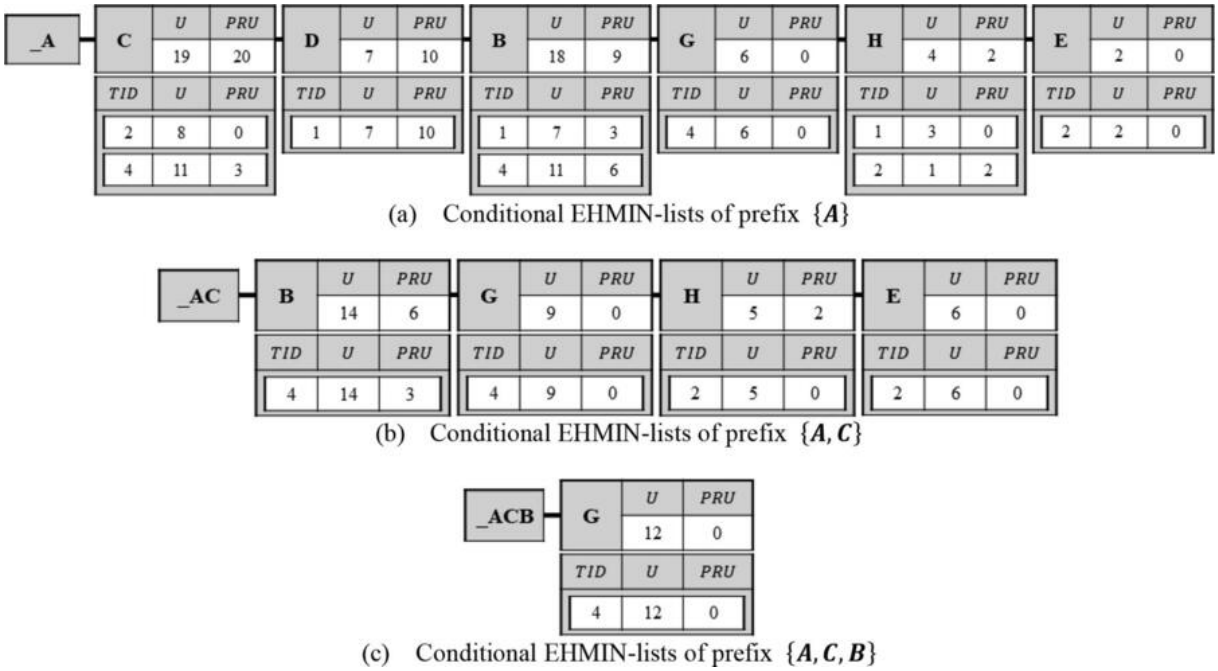


**Fig. 6.** Conditional EHMIN-lists.

vectors which are sorted according to the TID ascending order.

**Rational.** In order to construct the global EHMIN-lists, a database scan is conducted in the TID ascending order. In each transaction, information of the transaction is inserted into the TI-vectors of the global EHMIN-lists of the items in the transaction, so all the global EHMIN-lists have their TI-vectors, which are sorted in the TID ascending order. To construct 2-itemset conditional EHMIN-lists, the global EHMIN-lists are combined. In the combining process, the TI-vectors are traversed in order, and each entry of the transaction information is inserted into the TI-vectors of the conditional 2-itemset EHMIN-lists in order. The conditional EHMIN-lists for longer patterns than the 2-itemsets are constructed using the same method. Therefore, all the conditional EHMIN-lists also have their TI-vectors which are sorted by the TID ascending order. By using the proposition of Lemma 1, the comparing time for the construction of a new conditional EHMIN-list can be reduced.

Fig. 5 represents a search process(①~⑤) that finds the entries with a common TID from the two TI-vectors. All the TI-vectors in the global EHMIN-lists contain entries that are sorted according to TID ascending order, which is Lemma 1. The precondition is always valid, so the algorithm, which finds an intersection from the two sorted arrays, can also be applied to the TI-vectors. In Fig. 5, $\{A\}$ is the prefix, and it is combined with $\{C\}$. When noting the TIDs in the TI-vectors, $\{A\}$ is $\{1, 2, 4\}$, and $\{C\}$ is $\{2, 3, 4\}$. At first, the two iterators point to the first element in two TI-vectors. If one is pointing to the element with a smaller TID than the other one, it points to the next element. For example, the first elements of the TI-vectors of $\{A\}$ and $\{C\}$ are 1 and 2, and the element of $\{C\}$ is larger than the element of $\{A\}$. Therefore, only the TI-vector of $\{A\}$ moves to the next element. If the TIDs of the element that are pointed by iterators are equal, the utility (U) and the positive remaining utility (PRU) are calculated, and then they are inserted into the TI-vector in the new conditional EHMIN-list. Then, each iterator points to the next elements, respectively. Because the TI-vector of $\{A\}$ is moved to the second element, the TI-vectors of $\{A\}$ and $\{C\}$ represent the same element 2. Therefore, a new element is inserted into the TI-vector of $\{A, C\}$. A common TID 2 is inserted into the TID of the element, and 8, the sum of $U(A, T_2)$ and $U(C, T_2)$, is stored in $U(AC, T_2)$. The smaller value of $PRU(A, T_2)$ and $PRU(C, T_2)$, 0, is inserted into $PRU(AC, T_2)$. This is repeated until both iterators point to the last element. Assume that two TI-vectors have sizes that are $n$ and $m$. This search process then has a complexity of $\mathbf{O}(n + m)$, whereas a linear search has complexity of $\mathbf{O}(nm)$. If the total count of the combining processes is $N$, the total complexity are $\mathbf{O}(N(n+m))$ and $\mathbf{O}(Nnm)$. Therefore, it is predicted that this search process shows better runtime performance than a linear search on the databases, which have many transactions. In this paper, a method that adopts both this search process and the existing pruning techniques properly is presented. The pruning techniques that need to be considered in this search process are introduced in the following paragraphs.

*LA-Prune* is a pruning technique that is proposed in HUP-Miner (Krishnamoorthy, 2015). Assume there are two patterns, which are denoted by $X$ and $Y$. If $[U(X)+PRU(X)]-\sum_{X\subseteq T_k\wedge Y\not\subseteq T_k\wedge T_k\in DB}[U(X,T_k)+PRU(X,T_k)]\langle minutil$ is valid, $X\cup Y$ is a low-utility pattern. This was proofed as follows. Let the left terms of the above condition be $LA(X)$. The left terms can be expressed as $LA(X) = \sum_{X\cup Y\subseteq T_k\in DB}[U(X,T_k)+PRU(X,T_k)]$, so $LA(X)\geq\sum_{X\cup Y\subseteq T_k\in DB}[U(X\cup Y, T_k) + PRU(X\cup Y,T_k)] = U(X\cup Y) + PRU(X\cup Y)$. Thus, if $LA(X)\langle minutil$, $U(X\cup Y) + PRU(X\cup Y)\langle minutil$. According to *U-Prune*, $X\cup Y$ is then a low-utility pattern, and any super pattern of $X\cup Y$, which is denoted as $(X\cup Y)'$, is also a low-utility pattern. *LA-Prune* is applied to the process of traveling and comparing the TI-vectors. Before the process for combining $X$ and $Y$, a variable, $x$, is initialized as $U(X) + PRU(X)$. In the traveling process, the value of $U(X,T_k)+PRU(X,T_k)$ is subtracted from $x$ every transaction, $T_k$, that includes $X$ but not $Y$. The value of $x$ is checked at every update, and the current combining attempt is stopped if $x < minutil$.

*N-Prune* is a pruning technique that is proposed in the GHUM (Krishnamoorthy, 2018). Let a pattern be $X$ and any super pattern of $X$ be $X'$, and assume that $X'-X=Y\subseteq NI$ and $Y\neq\emptyset$. If $U(X)+\sum_{X\cup Y\subseteq T_k\in DB}U(Y,T_k)\langle minutil$, $X'$ is a low-utility pattern, and any super pattern of $X'$, $X''$, is also a low-utility pattern. The utility of $X'$ is $U(X') = \sum_{X\cup Y\subseteq T_k\in DB}U(X,T_k) + \sum_{X\cup Y\subseteq T_k\in DB}U(Y,T_k)$. In addition, $U(X) = \sum_{X\subseteq T_k\in DB}U(X,T_k)\geq\sum_{X\cup Y\subseteq T_k\in DB}U(X,T_k)$ is valid. Then, $U(X')\leq U(X)+\sum_{X\cup Y\subseteq T_k\in DB}U(Y,T_k)$ is also valid. The pattern growth is processed in the processing order, $X''-X'\subseteq NI$. It means $U(X'')\langle U(X')$ (*A-Prune*). Therefore, if $U(X)+\sum_{X\cup Y\subseteq T_k\in DB}U(Y,T_k)\langle minutil$, $X'$ and $X''$

are low-utility patterns ($\because \boldsymbol{U}(\boldsymbol{X}'')\langle \boldsymbol{U}(\boldsymbol{X}')\langle \boldsymbol{minutil}$). *N*-Prune is applied to the process of traveling and comparing TI-vectors, such as LA-Prune. Before the process of combining $\boldsymbol{X}$ and $\boldsymbol{Y}$, a variable, $\boldsymbol{y}$, is initialized as $\boldsymbol{U}(\boldsymbol{X})$. In the traveling process, the value of $\boldsymbol{U}(\boldsymbol{Y},\boldsymbol{T_k})\langle 0$ is subtracted from $\boldsymbol{y}$ for every transaction, $\boldsymbol{T_k}$, that includes both $\boldsymbol{X}$ and $\boldsymbol{Y}$. The value of $\boldsymbol{y}$ is also checked for every update, and the current combining attempt is stopped if $\boldsymbol{y} < \boldsymbol{minutil}$.

**Example 2.** This is a practical example for the construction process of the conditional EHMIN-lists. Fig. 6 shows a part of the mining process from Fig. 4(c). Firstly, Fig. 6(a) indicates a set of conditional EHMIN-lists for prefix $\{\mathbf{A}\}$ which are generated from Fig. 4(c). $\mathbf{U}(\{\mathbf{A}\}) + \mathbf{PRU}(\{\mathbf{A}\}) = 32 \geq 14$ (U-Prune), so the construction of the conditional EHMIN-lists for prefix $\{\mathbf{A}\}$ is conducted. The process of combining $\{\mathbf{A}\}$ with the other items after $\{\mathbf{A}\}$ is tried sequentially in the processing order. $\{\mathbf{A}\}$ and $\{\mathbf{C}\}$ emerged in the common transactions, which include $\mathbf{T}_2$ and $\mathbf{T}_4$. For $\mathbf{T}_2$, $\mathbf{U}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_2) = \mathbf{U}(\{\mathbf{A}\},\mathbf{T}_2) + \mathbf{U}(\{\mathbf{C}\},\mathbf{T}_2) = 4+4 = 8$ , and $\mathbf{PRU}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_2) = \mathbf{MIN}(\mathbf{PRU}(\{\mathbf{A}\},\mathbf{T}_2),\mathbf{PRU}(\{\mathbf{C}\},\mathbf{T}_2)) = 0$. In addition, for $\mathbf{T}_4$, $\mathbf{U}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_4) = 11$ and $\mathbf{PRU}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_4) = 17$, so $\mathbf{U}(\{\mathbf{A},\mathbf{C}\}) = 8+11 = 19$ and $\mathbf{PRU}(\{\mathbf{A},\mathbf{C}\}) = 0 + 17 = 17$, and the construction of the EHMIN-list for $\{\mathbf{A},\mathbf{C}\}$ is completely done. The other conditional EHMIN-lists for prefix $\{\mathbf{A}\}$ are also constructed in the same manner. In the conditional EHMIN-lists, $\{\mathbf{A},\mathbf{C}\}$ and $\{\mathbf{A},\mathbf{B}\}$ are mined as high-utility patterns. Next, Fig. 6(b) shows a set of conditional EHMIN-lists for the prefix $\{\mathbf{A},\mathbf{C}\}$ that is generated from Fig. 6(a). $\mathbf{U}(\{\mathbf{A},\mathbf{C}\}) + \mathbf{PRU}(\{\mathbf{A},\mathbf{C}\}) = 20 \geq 14$ (U-Prune), so the construction of the conditional EHMIN-lists for prefix $\{\mathbf{A},\mathbf{C}\}$ is conducted. $\{\mathbf{A},\mathbf{C}\}$ and $\{\mathbf{A},\mathbf{B}\}$ emerged only in a common transaction, $\mathbf{T}_4$. For $\mathbf{T}_4$, $\mathbf{U}(\{\mathbf{A},\mathbf{C},\mathbf{B}\},\mathbf{T}_4) = \mathbf{U}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_4) + \mathbf{U}(\{\mathbf{A},\mathbf{B}\},\mathbf{T}_4) - \mathbf{U}(\{\mathbf{A}\},\mathbf{T}_4) = 11+11-8 = 14$ and $\mathbf{PRU}(\{\mathbf{A},\mathbf{C},\mathbf{B}\},\mathbf{T}_4) = \mathbf{MIN}(\mathbf{PRU}(\{\mathbf{A},\mathbf{C}\},\mathbf{T}_4),\mathbf{PRU}(\{\mathbf{A},\mathbf{B}\},\mathbf{T}_4)) = 6$. The construction of the conditional EHMIN-lists for $\{\mathbf{A},\mathbf{C},\mathbf{B}\}$ is then done. In the same way, the other conditional EHMIN-lists for prefix $\{\mathbf{A},\mathbf{C}\}$ are also constructed. In the list, $\{\mathbf{A},\mathbf{C},\mathbf{B}\}$ is mined as a high-utility pattern. Finally, Fig. 6(c) represents a set of conditional EHMIN-lists for the prefix $\{\mathbf{A},\mathbf{C},\mathbf{B}\}$ that is generated from Fig. 6(b). Only an EHMIN-list for $\{\mathbf{A},\mathbf{C},\mathbf{B},\mathbf{G}\}$ is constructed, but $\{\mathbf{A},\mathbf{C},\mathbf{B},\mathbf{G}\}$ is not a high-utility pattern. The process goes back to the previous recursion depth, because there are no patterns that need to be combined in the current recursion depth.

**Lemma 2.** In the construction process of the conditional EHMIN-lists, accessibility to the transaction information of a prefix pattern of the current set of EHMIN-lists affects the construction performance enough.

**Rational.** In order to construct a conditional EHMIN-list, the transactions whose TIDs are equal are found from two given EHMIN-lists. In each common transaction, the utility of the conditional EHMIN-list is obtained by subtracting the utility of the common prefix pattern from the sum of the utilities of the two EHMIN-lists. The utility of the common prefix pattern should be searched at that time from the EHMIN-list of the common prefix pattern. This search operation is repeatedly required for all combinations of the current EHMIN-lists.

According to Lemma 2, an improved search technique for the utility information of the common prefix pattern should be considered. Thus, in this paper, a technique that holds the utility information in a temporary map container in advance rather than repeated searches, which is directly from the EHMIN-list of the common prefix EHMIN-list is presented. The details of this technique are subsequently introduced.

### 3.5. Algorithm descriptions of the proposed approach

We explain the proposed approach with its pseudo codes in detail in this section. The pseudo codes consist of a main procedure and two sub procedures. We first describe the main procedure in order to explain the entire process of the EHMIN, and we then explain the sub procedures, which are mentioned in the main procedure but are not explained in detail. This algorithm explanation helps express the suggested approach in computer languages, which properly applies the strategies that are mentioned. Fig. 7 shows the main procedure of the EHMIN. Once a transaction database with utility information and a user-defined threshold are given, the EHMIN algorithm outputs a set of high-utility patterns and performance results. As mentioned in Section 3.2, the overall process can be divided into 3 steps, which include *1st Database Scan*, *2nd Database Scan*, and *Mining*. Fig. 7 shows the 3 steps, which are separated by dotted lines. Firstly, the first step, *1st Database Scan*, is described in Lines 01–10. The purpose of this step is to prepare the construction of data structures by collecting information about a given database. (Line 01) The external utility data of all the items in a given database is loaded. The external utility data is required in order to calculate the utility of each item during the database scans. (Lines 02–07) While each transaction is being scanned, the positive transaction utility (PTU) of each transaction and the positive transaction weighted utility (PTWU) of each item are calculated.

Each PTU can be calculated as a sum of the positive utilities of the items in the transaction, and each PTWU can be calculated by accumulating the PTU of each transaction where the item appears. (Line 08) The total PTU and the given threshold (***rate***) are multiplied in order to obtain the minimum utility (***minUtil***). (Line 10) The processing order is decided by the PTWU and the support information of all the items, and all items are dealt with by the ordering numbers in the remain processes. Next, the second step, *2nd Database Scan*, is described in Lines 11–34. The purpose of this step is to construct the data structures and prepare the mining process. (Lines 11–34) The given database is scanned one more time. (Lines 12–17) *PTWU-Prune* is applied by using the PTWU information that is obtained in the previous step. The PTU information is re-calculated, because the pruned items should be not considered anymore. (Lines 18–24) The items and utility information are stored temporarily and then sorted in the processing order. (Lines 25–32) The global EHMIN-lists are subsequently constructed by visiting the stored items and utility information in the reversed order. The reason for the reversed order is to calculate the remaining utilities. The remaining

```
Input :    An transactional database, DB = {T1, T2, ..., Tm}
           A given threshold, rate
Output :   A set of High-Utility Pattern results, HUP (global)
Variable : A set of global EHMIN-lists, UL
           A minimum utility, minUtil (global)

EHMIN Algorithm
01. load the external utility data                                  // Step 1 :: 1st Database Scan
02. for each transaction, Tk, in DB do
03.     for each item, i, and internal utility, IU(i), in Tk do
04.         calculate PTWU and support of distinct items
05.     end for
06.     calculate PTU
07. end for
08. calculate minUtil with the total PTU and rate
09. initialize the set of global EHMIN-lists for 1-itemsets, UL
10. index all distinct items by processing order using PTWU and support information
11. for each transaction, Tk, in DB do                              // Step 2 :: 2nd Database Scan
12.     PTUk ← 0    // re-compute PTU
13.     for each item, i, and internal utility, IU(i), in Tk do
14.         if (PTWU(i) > minUtil) then    // PTWU-Prune
15.             PTUk ← PTUk + PU(i,Tk)
16.         end if
17.     end for
18.     initialize a temporary map, tmp
19.     for each item, i, and internal utility, IU(i), in Tk do
20.         tmp.insert(<index(i), IU(i) × EU(i)>)
21.         if (PTWU(i) > minUtil) then
22.             newPTWU(i) ← newPTWU(i) + PTUk      // re-compute PTWU
23.         end if
24.     end for
25.     rutil ← 0
26.     for each index of item, idx, and utility, U(i), in tmp.reversed() do
27.         Ui ← UL.find_or_create(idx)
28.         Ui.TI_vector.insert(< Tk.id, U(i), rutil >)
29.         if (U(i) > 0) then
30.             rutil ← rutil + U(i)
31.         end if
32.     end for
33.     calculate EUCS[∀ik, ∀jk] with PTUk
34. end for
34. HUP ← ∅                                                         // Step 3 :: Mining
35. call EHMIN_Mine(∅, UL, ∅)
36. return HUP
```

**Fig. 7.** Main procedure: EHMIN.

**Input :** A prefix EHMIN-list, $P$
A set of EHMIN-lists, $UL$
A prefix itemset, $pref$
**Output :** None
**Variable :** A set of conditional EHMIN-lists, $CL$

*EHMIN_Mine Algorithm*

```
01. initialize the prefix utility map, pfutils
02. for each s in P do
03.     pfutils.insert(< s.tid, s.U >)
04. end for
05. for each U_k in UL do
06.     if (U_k.U ≥ minUtil) then
07.         HUP.insert(pref ∪ {U_k.item})
08.     end if
09.     if (U_k.U + U_k.PRU ≥ minUtil) then      // U-Prune
10.         initialize the set of conditional EHMIN-lists, CL
11.         for each U_l in UL do   //k < l
12.             if (EUCS[U_k.item, U_l.item] ≥ minUtil) then      // EUCS-Prune
13.                 C ← EHMIN_Combine(U_k, U_l, pfutils)
14.                 if (C ≠ ∅) then
15.                     CL.insert(C)
16.                 end if
17.             end if
18.         end for
19.         if (|CL| > 0) then
20.             call EHMIN_Mine(U_k, CL, pref ∪ {U_k.item})
21.         end if
22.     end if
23. end for
```

**Fig. 8.** Sub-procedure 1: EHMIN_Mine.

pattern is a set of items after a pattern in a transaction (Definition 15). Thus, the remaining utility can be calculated by accumulating the utility of each item in the transaction that is sorted in the reversed order (Line 33). The PTU of the current transaction is added to the EUCS values for all pairs of items in the current transaction. The EUCS are completely prepared at the end of this step because all the values are calculated by accumulating the PTUs. Finally, the third step, which is *Mining*, is described in Line 34–36. In this step, all the valid patterns are mined using the data structures that are prepared in the previous step. (Line 34) The set of result patterns is initialized. (Line 35) A sub procedure, which is *EHMIN_Mine*, is called by taking the set of global EHMIN-lists as its parameter. All the mined patterns in this procedure are then stored in the set of result patterns, which is a global variable. The details of the procedure are described in the explanation of sub-procedure 1. (Line 36) When the process of *EHMIN_Mine* is all completed, the main procedure outputs the set of the result patterns, and it is then exited.

Fig. 8 represents the pseudo code of the first sub procedure of the EHMIN algorithm, *EHMIN_Mine*. It checks the pattern information in a set of EHMIN-lists, one of the parameters, and extracts high-utility patterns. After that, it combines the current EHMIN-lists and recursively calls itself in order to check the pattern information in the conditional EHMIN-lists. (Lines 01–04) The TID and the utility information of the prefix pattern is held in a map container from one of the parameters, which is the EHMIN-list for the prefix pattern, because this information is required many times to calculate the utility information of combined patterns (Lemma 2). The total runtime can be reduced by making them faster to be obtained in advance. (Lines 05–23) The EHMIN-lists given as one of the parameters are visited in the processing order. (Lines 06–08) If the pattern of the visited EHMIN-list is a high-utility pattern, the pattern is added to the set of result patterns. (Lines 09–22) *U-Prune* checks whether the pattern is promising or not. If it is promising, it becomes a prefix pattern. (Lines 11–18) In order to try to combine it with the other patterns in the given EHMIN-lists, the remaining EHMIN-lists are all visited. (Lines 12–17) By *EUCS-Prune*, whether two patterns of the visited EHMIN-lists deserve to be combined is checked. (Line 13) If they deserve to be combined, a sub procedure, *EHMIN_Combine*, is called by taking them as the parameters. A new EHMIN-list for the combined pattern is then returned from the procedure. The details are described in the explanation of sub-procedure 2. (Lines 14–16) The procedure returns not NULL(∅) but an EHMIN-list if

**Input :** Two EHMIN-lists, $U_k, U_l$
The prefix utility map, $pfutils$
**Output :** A new conditional EHMIN-list, $C$
**Variable :** None

*EHMIN_Combine Algorithm*

```
01. initialize the conditional utility pattern list, C
02. C.item ← U_l.item
03. x ← U_k.U + U_k.PRU
04. y ← U_k.U
05. s_k ← U_k.TI_vector.begin()
06. s_l ← U_l.TI_vector.begin()
07. while (s_k ≠ U_k.TI_vector.end() and s_l ≠ U_l.TI_vector.end()) do
08.     if (s_k.tid = s_l.tid) then
09.         if (pfutil ≠ ∅) then
10.             pfutil ← pfutils.find(s_k.tid)
11.         else then
12.             pfutil ← 0
13.         end if
14.         util ← s_k.U + s_l.U − pfutil
15.         rutil ← MIN(s_k.PRU, s_l.PRU)
16.         C.TI_vector.insert(< s_k.tid, util, rutil >)
17.         y ← y + (s_l.U − pfutil)
18.         if (s_k.PRU = 0 and y < minUtil) then      // N-Prune
19.             return ∅
20.         end if
21.         s_k ← s_k.next()
22.         s_l ← s_l.next()
23.     else if (s_k.tid > s_l.tid) then
24.         s_l ← s_l.next()
25.     else then   // s_k.tid < s_l.tid
26.         x ← x − (s_k.U + s_k.PRU)
27.         if (x < minUtil) then      // LA-Prune
28.             return ∅
29.         end if
30.         s_k ← s_k.next()
31.     end if
32. end while
33. if (C.TI_vector.isEmpty()) then
34.     return ∅
35. end if
36. return C
```

**Fig. 9.** Sub-procedure 2: EHMIN_Combine.

the patterns can be combined successfully. If the return is not NULL(∅), it is inserted into a set for the new EHMIN-lists. (Line 19) If the size of the set of new EHMIN-lists of the current prefix is not<1, *EHMIN_Mine* is recursively called by taking the set and the prefix pattern information.

Fig. 9 indicates the pseudo code of the second sub-procedure of the EHMIN algorithm, which is *EHMIN_Combine*. It takes two EHMIN-lists whose patterns are combined and outputs a new EHMIN-list for the combined pattern. If the EHMIN-lists are not promising to be combined or cannot be combined, it returns NULL(∅). (Line 01) A new EHMIN-list is generated and initialized. (Lines 03–04) Two variables for *LA-Prune* and *N-Prune* are prepared. The initial values are set for their techniques. (Lines 05–32) Two TI-vectors in the EHMIN-lists, which are given as the parameters, are visited. This search method, which is based on an algorithm that finds an intersection of two sorted vectors, can be applied because of Lemma 1. Once the two TI-vectors whose sizes are $n$ and $m$ are given, this search method has complexity of not $\mathbf{O}(nm)$ but $\mathbf{O}(n + m)$. The operation of searching a pair of TI-vectors is a big part of the total mining operation, so this efficient search method contributes in regards to improving the performance of the algorithm. (Lines 08–22) If the common TID is found, the new transaction information is calculated and inserted into the TI-vector in the new EHMIN-list. (Lines 09–13) If the given two patterns have a common prefix pattern, the utility of the prefix pattern of them is obtained to remove the overlapped values from the sum of the utilities of them. If they don't have, the utility of the combined pattern is just the sum of the utilities of them. (Lines 14–16) The utility and the remaining utility of the combined pattern are calculated, and the new transaction information is then inserted into the TI-vector of the new EHMIN-list. (Lines 17–20) After that, the variable of

*N-Prune* is updated, and the condition of *N-Prune* is checked. If the pruning condition is satisfied, this sub procedure returns NULL(∅) and is terminated because the combined pattern is not a high-utility pattern and an unpromising pattern. (Lines 26–29) Moreover, the variable of *LA-Prune* is updated, and the condition of *LA-Prune* is checked. If the pruning condition is satisfied, this sub procedure returns NULL(∅) and is terminated because the combined pattern is not a high-utility pattern and an unpromising pattern. (Lines 33–35) The pruning conditions are not satisfied, so this sub-procedure also returns NULL(∅) if there are no entries in the TI-vector of the new EHMIN-list. (Line 36) This sub-procedure finally returns the new EHMIN-list only if all the combination nonconformity conditions are avoided.

### 3.6. Complexity analysis

In this section, we show the complexity analysis of the proposed algorithm and one of the state-of-the-art algorithms, GHUM (Krishnamoorthy, 2018). Both algorithms can be divided into 2 phases, respectively: *Database scan* and *Mining*. Assume that a given transactional database consists of $m$ transactions and $n$ items. This analysis of complexity considers $m$ and $n$ as basic variables.

In the first phase, the algorithms scan the given database 2 times. In the first scan, they load the external utility data and traverse all the transactions in order to calculate the positive transaction utilities (PTUs) for all the transactions, the positive transaction weighted utilities (PTWUs) of all the items, and the supports of all items. In addition, the processing order is decided by the PTWU and support information of all the items. Thus, both of them have the complexity of $\mathbf{O}(n+mn+n) = \mathbf{O}(mn)$ in the first scan. They subsequently conduct the second scan in order to construct their global list structures. The number of lists is equal to $n$, and the lists are sorted in the processing order. Assume that the quick sort manner is used for it. In the sorting process, both of them have the complexity of $\mathbf{O}(n \bullet \mathbf{log}_2 n)$. In order to hold the transactions information in each list, the proposed algorithm uses vector containers, and the GHUM uses map containers. It is generally known that vector containers have the complexity of $\mathbf{O}(1)$ in one insertion. Thus, the proposed algorithm has the complexity of $\mathbf{O}((m \bullet 1) \bullet n) = \mathbf{O}(mn)$ in the total insertion of the transaction information. Meanwhile, it is generally known that map containers have the complexity of $\mathbf{O}(\mathbf{log}_2 h)$ in one insertion where $h$ is a height of a binary tree in a map container. Thus, GHUM has complexity of $\mathbf{O}\left(\left(m \bullet \left(\sum_{k=1}^{n} \mathbf{log}_2 k\right)\right) \bullet n\right)$ in the total insertion of the transaction information. Using a fact that $\sum_{k=1}^{n} \mathbf{log}_2 k \approx \int_1^n \mathbf{log}_2 k dk = \left[k \bullet \mathbf{log}_2 k - \left(\frac{k}{\ln 2}\right)\right]_1^n = n \bullet \mathbf{log}_2 n - ((n-1)/\ln 2)$, the complexity can be considered as $\mathbf{O}((m \bullet (n \bullet \mathbf{log}_2 n - n)) \bullet n) = \mathbf{O}\left(mn^2 \bullet \mathbf{log}_2 n - mn^2\right)$ just for a simple comparison.

In the second phase, the worst-case scenario is that all the items are never pruned. Let $HUP_k$ be a set of high-utility patterns whose length is $k$. There can be $HUP_1$, $HUP_2$, $HUP_3$, …, and $HUP_n$ because of $n$. In addition, all sizes of the pattern sets are $|HUP_1| = (n,1)$, $|HUP_2| = (n,2)$, $|HUP_3| = (n,3)$, …, and $|HUP_n| = (n,n)$. Therefore, the total number of conditional lists is $\sum_{k=2}^{n} |HUP_k| = \sum_{k=2}^{n} (n,k) = 2^n - (n+1)$. As mentioned in Section 3.4, if the total count of combining processes is $N$, the total complexity of EHMIN is $\mathbf{O}(N(m+m)) = \mathbf{O}(Nm)$, and the total complexity of GHUM is $\mathbf{O}(Nm^2)$. Because $N = 2^n - (n+1)$, the complexity of the proposed algorithm is $\mathbf{O}(m \bullet 2^n - mn)$, and the complexity of GHUM $\mathbf{O}\left(m^2 \bullet 2^n - m^2 n\right)$ in this phase.

Finally, the algorithms have different complexities in only two operations, which include the insertion of transaction information and the construction of conditional lists. In the first operation, the complexity of the proposed algorithm is $\mathbf{O}(mn)$, and the complexity of GHUM is $\mathbf{O}\left(mn^2 \bullet \mathbf{log}_2 n - mn^2\right)$. The proposed algorithm has $(n \bullet \mathbf{log}_2 n - n)$ better complexity than the GHUM. Moreover, in the other one, the complexity of the proposed algorithm is $\mathbf{O}(m \bullet 2^n - mn)$, and the complexity of the GHUM is $\mathbf{O}\left(m^2 \bullet 2^n - m^2 n\right)$. The proposed algorithm has $m$ better complexity than the GHUM. The total complexity of the proposed algorithm is $\mathbf{O}(mn + n \bullet \mathbf{log}_2 n + mn + (m \bullet 2^n - mn))$, and the total complexity of the GHUM is $\mathbf{O}\left(mn + n \bullet \mathbf{log}_2 n + \left(mn^2 \bullet \mathbf{log}_2 n - mn^2\right) + \left(m^2 \bullet 2^n - m^2 n\right)\right)$.

**Table 5**
Features of the real datasets.

| Dataset | Num. Of Trans. | Num. of Items | Avg. Trans. Size | Data Size | Type |
|---|---|---|---|---|---|
| Chess | 3,196 | 75 | 37 | 0.32 MB | Dense, Real |
| Connect | 67,557 | 129 | 43 | 8.82 MB | |
| Mushroom | 8,124 | 119 | 23 | 0.54 MB | |
| Pumsb | 49,046 | 468 | 74 | 15.9 MB | |
| Kosarak | 990,002 | 41,270 | 8.1 | 31.4 MB | Sparse, Real |
| Chainstore | 1,112,949 | 46,085 | 7.2 | 45.5 MB | |
| Retail | 88,162 | 16,469 | 10.3 | 3.97 MB | |
| BMS-POS | 515,597 | 1,656 | 6.5 | 11.3 MB | |

## 4. Performance evaluation

### 4.1. Experimental environment and datasets

In this section, we present the various experiments by comparing the proposed approach, which is the EHMIN, with the existing negative utility pattern mining approaches, which include GHUM (Krishnamoorthy, 2018) and FHN (Lin et al., 2016). GHUM and FHN are list-based algorithms. All the algorithms that are used for performance evaluation extract negative utility patterns, but there are performance differences, which are due to the pattern expansion process and the pruning method. The GHUM applied improved pruning technology compared to a pruning technique that is used by the FHN. EMINEM uses an improved pruning method and novel pattern expansion technique. All of them were implemented in C/C++ languages and complied via Visual studio 2017 on a computer with an Intel Core i6-6700 K CPU @ 4.00 GHz, 32 GB RAM, and a Windows 10 OS.

In order to evaluate the performance of the approaches on various datasets, both real and synthetic datasets are used as the input data. Table 5 shows a list of real datasets from the FIMI repository (https://fimi.ua.ac.be/data/). They can be separated into dense datasets and sparse datasets, which depends on their features. The dense data set consists of long transactions and a small number of items. On the other hand, sparse data sets have short transactions and a large number of items. They were used to evaluate the runtime and memory usage performance on various thresholds. Meanwhile, Table 6 provides a list of synthetic datasets from an IBM data generator. They are also separated

**Table 6**
Characteristics of the synthetic datasets.

| Dataset | Num. Of Trans. | Num. of Items | Avg. Trans. Size |
|---|---|---|---|
| T10I4D200K | 200,000 | 1,000 (fixed) | 10 (fixed) |
| T10I4D400K | 400,000 | | |
| T10I4D600K | 600,000 | | |
| T10I4D800K | 800,000 | | |
| T10I4D1000K | 1,000,000 | | |
| T10I4D2000K | 2,000,000 | | |
| T10I4D3000K | 3,000,000 | | |
| T10N10000L1000 | 100,000 (fixed) | 10,000 | 10 |
| T20N20000L2000 | | 20,000 | 20 |
| T30N30000L3000 | | 30,000 | 30 |
| T40N40000L4000 | | 40,000 | 40 |

**Table 7**
Statistical characteristics of the utility settings.

| | Utiltiy | Average | variance | Standard Deviation | Minimum | Maximum | Ratio of Negative |
|---|---|---|---|---|---|---|---|
| Chess | Internal | 5.505 | 8.308 | 2.882 | 1 | 10 | 0 % |
| | External | 1.003 | 19.412 | 4.406 | −8.168 | 8.196 | 33.3 % |
| Connect | Internal | 5.5 | 24.5 | 4.950 | 1 | 10 | 0 % |
| | External | 0.875 | 18.701 | 4.324 | −9.454 | 9.649 | 37.2 % |
| Mushroom | Internal | 5.505 | 8.245 | 2.871 | 1 | 10 | 0 % |
| | External | 0.879 | 18.344 | 4.283 | −9.007 | 9.649 | 37 % |
| Accidents | Internal | 5.5 | 40.5 | 6.364 | 1 | 10 | 0 % |
| | External | 1.02 | 17.268 | 4.155 | −9.454 | 9.649 | 35.7 |
| Pumsb | Internal | 5.5 | 4.5 | 2.121 | 1 | 10 | 0 % |
| | External | 1.181 | 18.283 | 4.276 | −9.968 | 9.978 | 34.4 |
| Kosarak | Internal | 4 | 8 | 2.828 | 1 | 10 | 0 % |
| | External | 1.242 | 17.951 | 4.237 | −9.999 | 9.997 | 33.5 % |
| Chainstore | Internal | 1 | 0 | 0 | 1 | 370 | 0 % |
| | External | 1.342 | 41.646 | 6.453 | −222 | 180 | 33.6 % |
| Retail | Internal | 5.502 | 8.249 | 2.872 | 1 | 10 | 0 % |
| | Internal | 2.999 | 2.0 | 1.414 | 1 | 5 | 0 % |
| | Internal | 50.431 | 823.802 | 28.858 | 1 | 100 | 0 % |
| | External | 1.253 | 18.096 | 4.254 | −9.989 | 9.993 | 33.4 % |
| Foodmart | Internal | 3.099 | 0.688 | 0.829 | 1 | 6 | 0 % |
| | External | 0.751 | 4.791 | 2.189 | −3.980 | 3.980 | 33.9 % |
| Bms-pos | Internal | 4.5 | 4.5 | 2.121 | 1 | 10 | 0 % |
| | External | 1.965 | 33.189 | 5.761 | −9.986 | 9.994 | 32.8 % |
| T10I4DxK | Internal | 5.504 | 8.244 | 2.871 | 1 | 10 | 0 % |
| | External | 1.377 | 29.117 | 5.396 | −29.990 | 30.990 | 33.4 % |
| TxNyLz | Internal | 5 | 2 | 1.414 | 1 | 10 | 0 % |
| | External | 1.374 | 38.277 | 6.187 | −168.020 | 222 | 33.4 % |

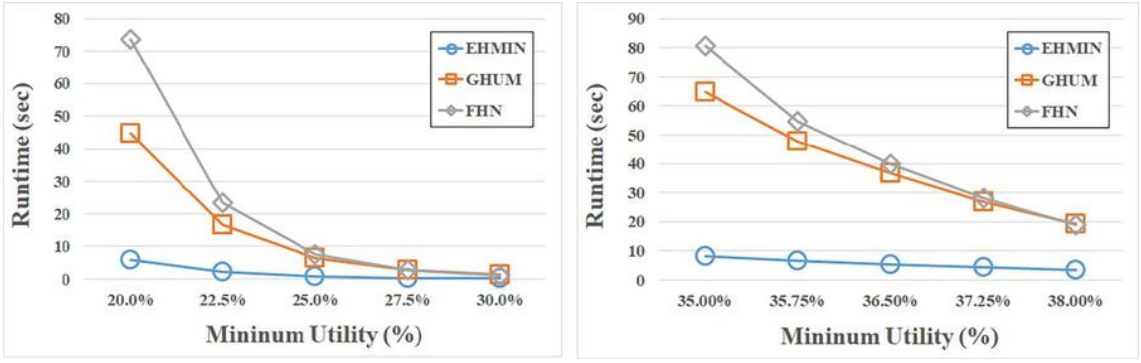


**Fig. 10.** Runtime tests on the Chess and Connect datasets.

into two groups. One group contains datasets that have the constant number of items, the constant average length of transactions, but the different number of transactions. The other group contains datasets that have a constant number of transactions, but a different number of items, and a different average length of transactions. The groups are used to perform the scalability tests.

Table 7 represents the statistical characteristics of the datasets that were used in the performance evaluations. The internal utility and the external utility of the datasets except for the chainstore dataset were randomly generated using a log-normal distribution manner, which is described in the previous studies (Krishnamoorthy, 2018; Lin et al., 2016). In the rows of external utilities, the range of external utility of items with negative or positive utility is specified to proceed with performance evaluation considering negative utility. The chainstore dataset

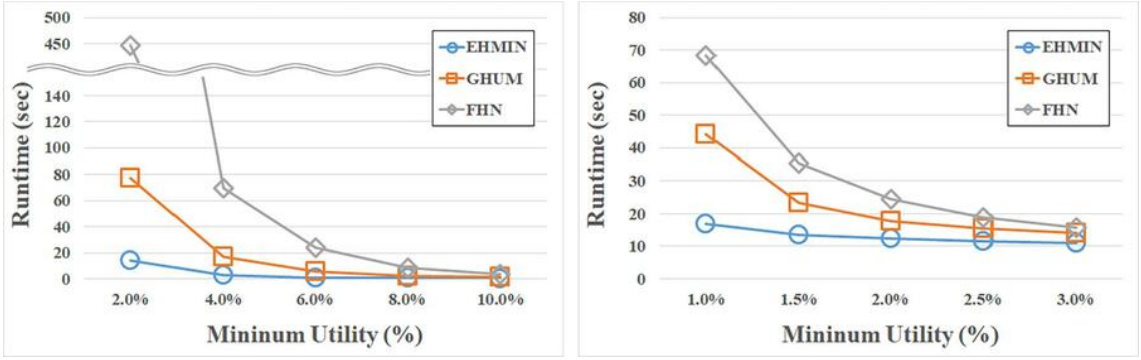


**Fig. 11.** Runtime tests on the Chess and Connect datasets.

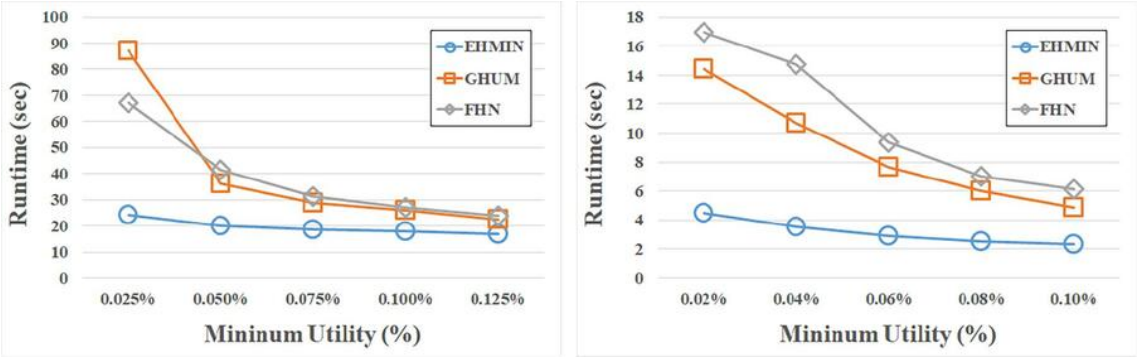


**Fig. 12.** Runtime tests on the Mushroom and Kosarak datasets.

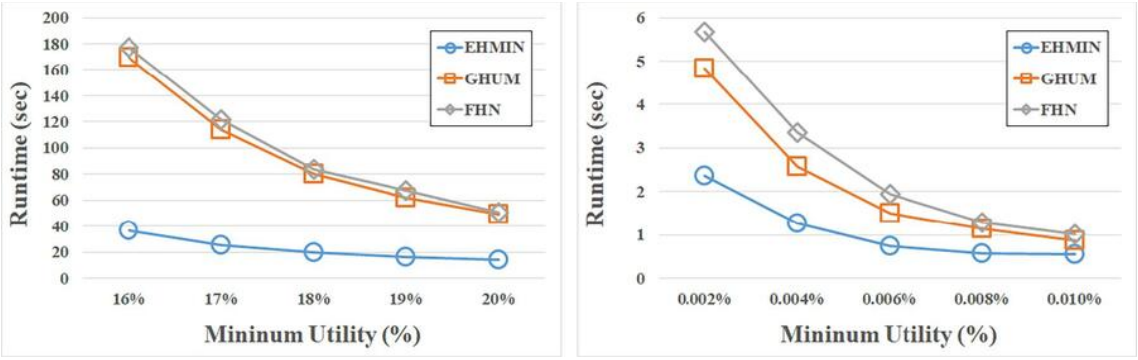


**Fig. 13.** Runtime tests on the Mushroom and Kosarak datasets.

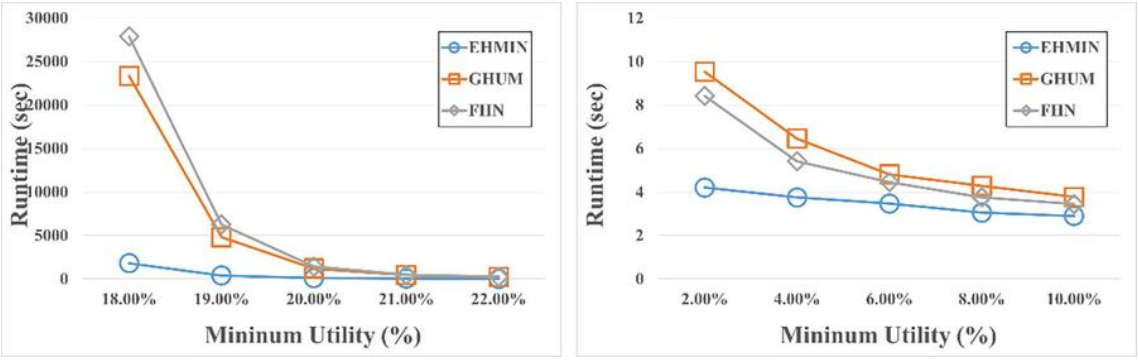


**Fig. 14.** Runtime tests on the Chainstore and Retail datasets.

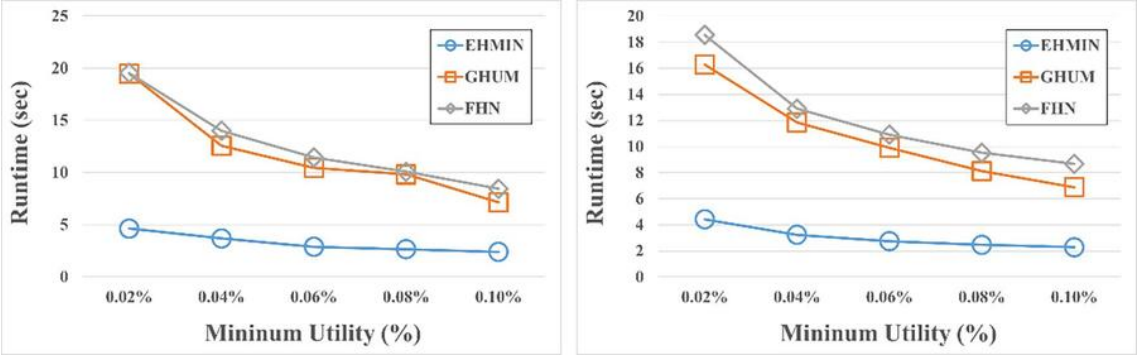


**Fig. 15.** Runtime tests on the Chainstore and Retail datasets.

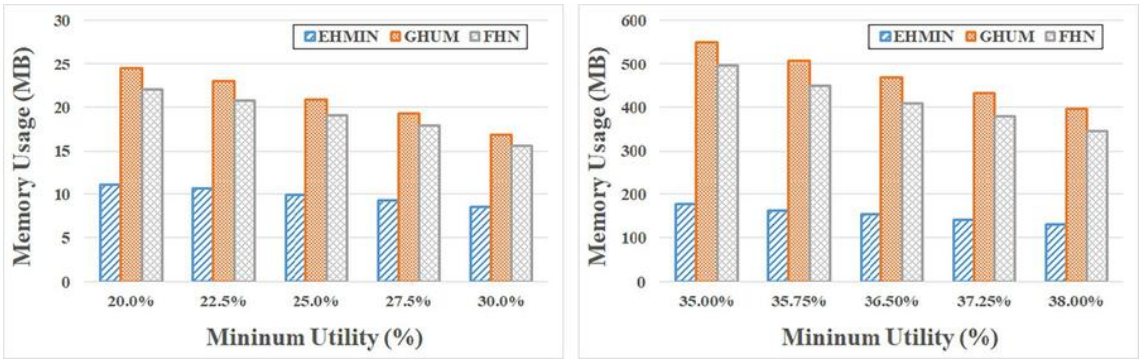


**Fig. 16.** Runtime tests on the Accidents and Foodmart datasets.

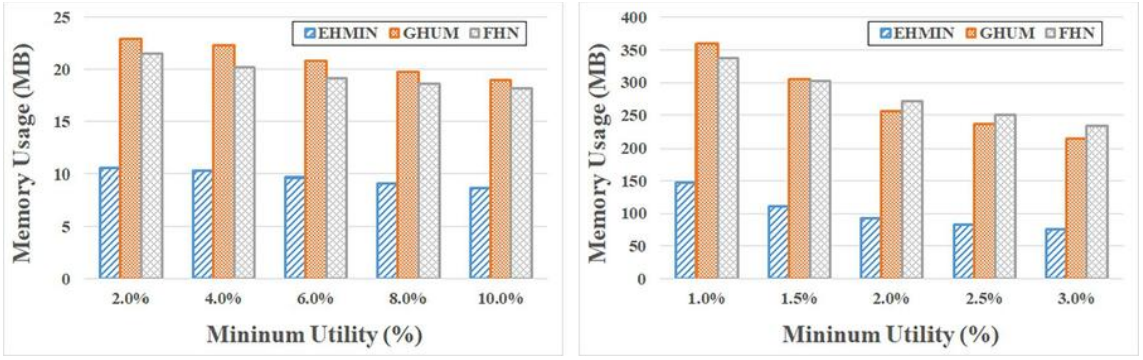


**Fig. 17.** Runtime tests on the Accidents and Foodmart datasets.

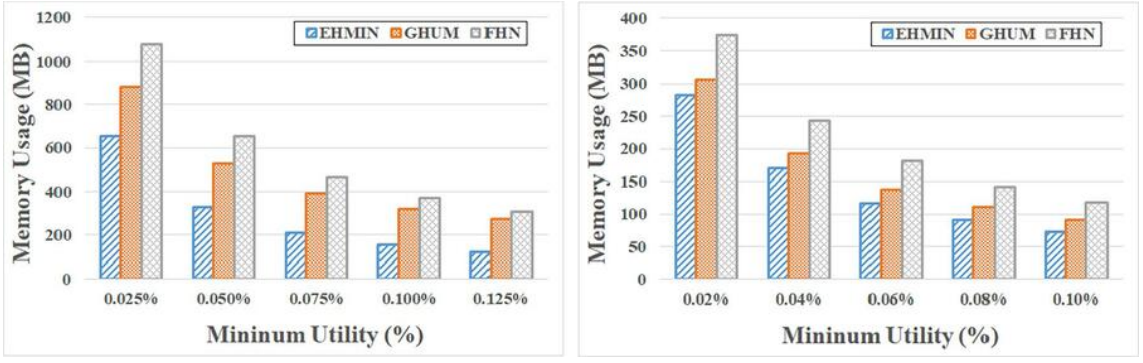


**Fig. 18.** Runtime tests on the Pumsb and Bms-pos datasets.

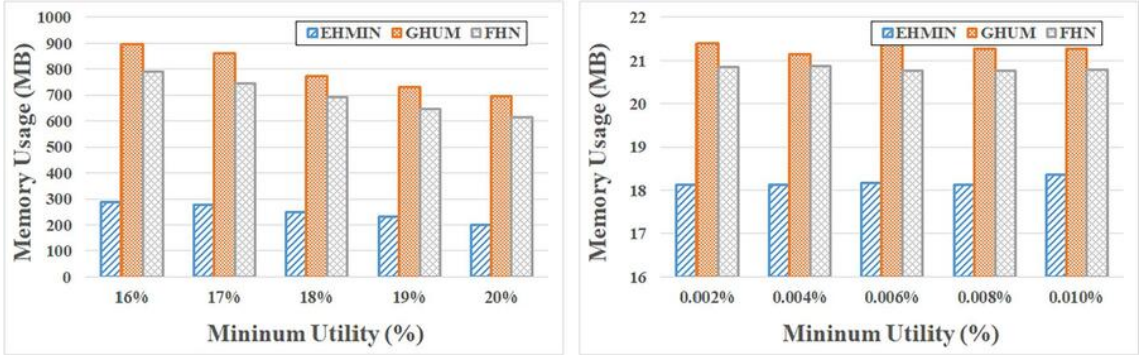


**Fig. 19.** Runtime tests on the Pumsb and Bms-pos datasets.

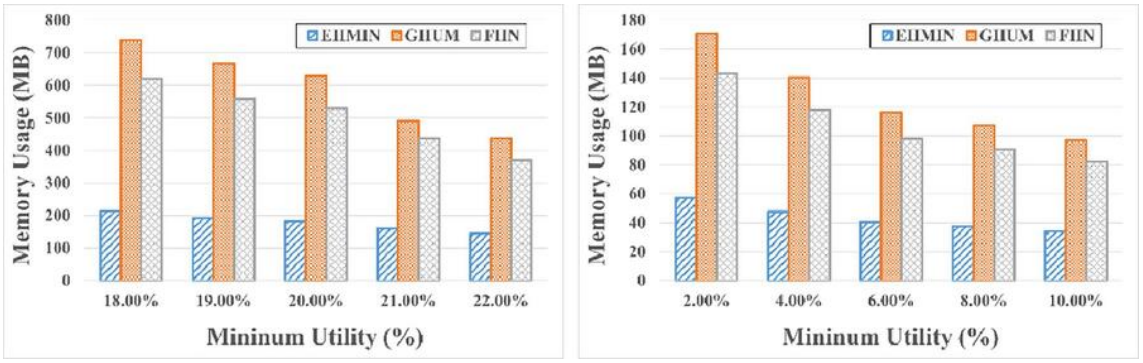


**Fig. 20.** Runtime tests by changing internal utility.

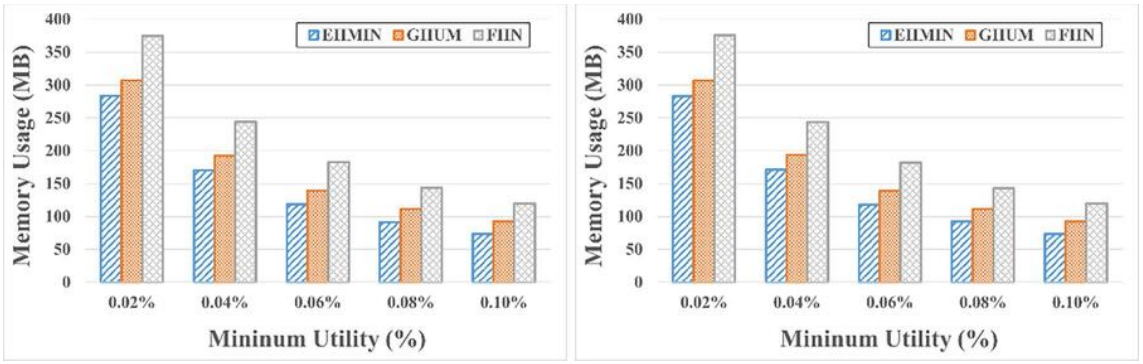


**Fig. 21.** Runtime tests by changing internal utility.

has real internal and external utilities, so the existing utility values are applied. In addition, the performance of the algorithms can vary depending on the internal utility settings. Therefore, for fair performance evaluations, the retail dataset was used in the scenario example, and it was evaluated in various internal utility settings.

### 4.2. Analysis of the runtime test

The results of the runtime performances of the algorithms are shown and analyzed in this section. In all the tests, the threshold settings were decided, so the number of result patterns is different on each threshold. Figs. 10–21 are line graphs that show the results of the runtime performances on the real datasets. The threshold constantly increases in each dataset. If the threshold, which is ***minUtil***, decreases, the runtime generally increases. The reason is that the number of patterns that need to be processed increases. In each dataset, all the algorithms suffer this type of phenomenon, so the differences between them illustrate their performance.

Fig. 10 represents the runtime results on the Chess dataset, where the threshold increases from 20.0 % to 30.0 %. When the threshold decreases, the runtime of the EHMIN increases slowly, whereas the runtimes of the others become quicker. Where the threshold is 20.0 %, the EHMIN has a 7.45x better runtime performance than the GHUM and a 12.22x better runtime performance than the FHN. The gaps become larger as the threshold decreases. Thus, the EHMIN has a better runtime performance than others on the Chess datasets. Next, Fig. 11 shows the runtime results on the Connect dataset where the threshold increases from 35.00 % to 38.00 %. The runtime results of the algorithms increase in a similar aspect. Where the threshold is 35.00 %, the EHMIN has a 7.88x better runtime performance than the GHUM and a 9.78x better runtime performance than the FHN. Next, Fig. 12 illustrates the runtime results on the Mushroom dataset where the threshold increases from 2.0 % to 10.0 %. The runtime results of the EHMIN and GHUM increase in a similar manner, but the runtime of the FHN increases very fast. The main reason is that the data structure of the FHN has more data fields. When the threshold is 2.0 %, the EHMIN has a 5.42x better runtime performance than the GHUM and a 31.32x better runtime performance than the FHN. Next, Fig. 13 shows the runtime results on the Kosarak dataset where the threshold increases from 1.0 % to 3.0 %. The runtime results of the algorithms increase in the similar shape. Where the threshold is 1.0 %, the EHMIN has about a 2.63x better runtime performance than the GHUM and a 4.07x better runtime performance than the FHN. Next, Fig. 14 represents runtime results on the Chainstore dataset where the threshold increases from 0.025 % to 0.125 %. The runtime results of the algorithms increase in a similar aspect. Where the threshold is 0.025 %, the EHMIN has about a 3.59x better runtime performance than the GHUM and a 2.76x better runtime performance than the FHN. Next, Fig. 15 shows runtime results on Retail dataset where the threshold increases from 0.02 % to 0.10 %. The runtime results of the algorithms increase in a similar manner. Where the threshold is 0.02 %, EHMIN has about 3.20x and 3.76x better runtime performance than GHUM and FHN, respectively. Fig. 16 illustrates the runtime results on the Accidents dataset where the threshold increases from 16 % to 20 %, and Fig. 17 represents the runtime results on the Foodmart dataset where the threshold increases from 0.002 % to 0.010 %. Fig. 18 and Fig. 19 show the runtime results in the Pumsb and Bms-pos datasets. The thresholds are set from 2 % to 10 % and from 18 % to 22 %. Finally, Fig. 20 and Fig. 21 represent the runtime results that were measured by changing the internal utility setting on the Retail dataset used in the scenario example. The internal utility ranges are changed to 1–5 and 1–100 while maintaining the threshold setting. On all real datasets and utility settings, the runtime of EHMIN has the slowest increase as the threshold decreases. Moreover, the gaps become larger as the threshold decreases on all the datasets. It is because EHMIN has an improved pattern expansion process than the other algorithms. The pattern expansion is the most repeated operation during the mining process. Therefore,

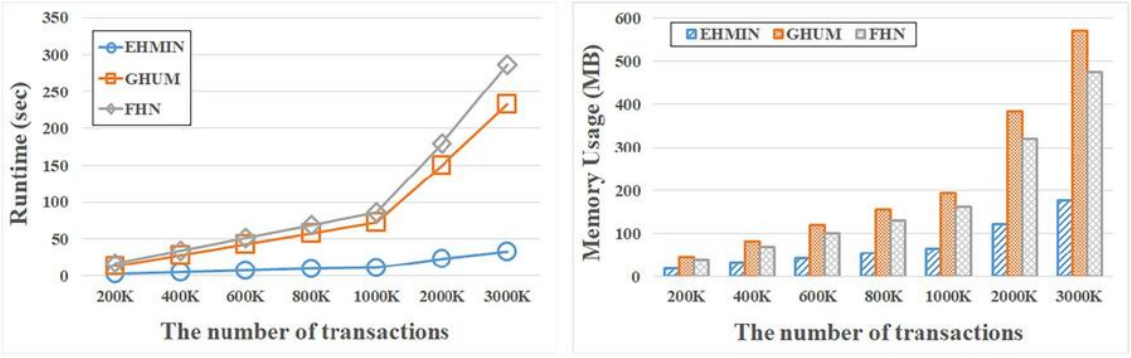


**Fig. 22.** Memory usage tests on the Chess and Connect datasets.

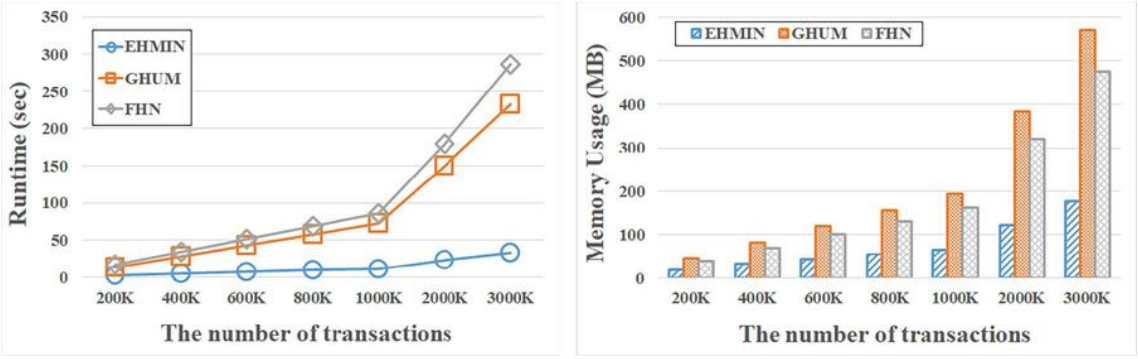


**Fig. 23.** Memory usage tests on the Chess and Connect datasets.

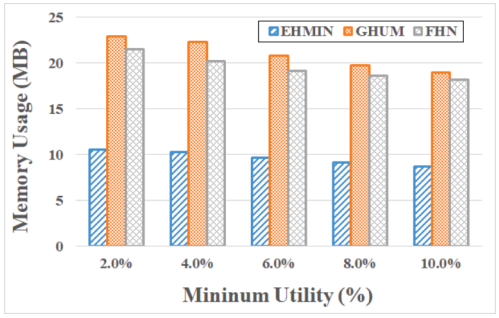


**Fig. 24.** Memory usage tests on and Mushroom and Kosarak datasets.

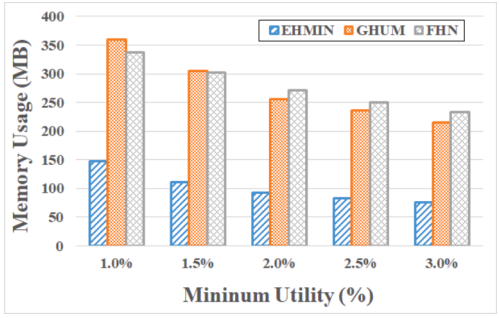


**Fig. 25.** Memory usage tests on and Mushroom and Kosarak datasets.

improvements in the pattern extension process represent dramatic performance advancement. In conclusion, the EHMIN always showed the best performance regardless of the dataset, which is due to the improved pattern expansion algorithm, and the performance evaluation and

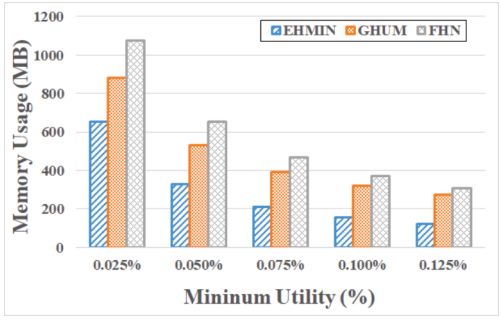


**Fig. 26.** Memory usage tests on the Chainstore and Retail datasets.

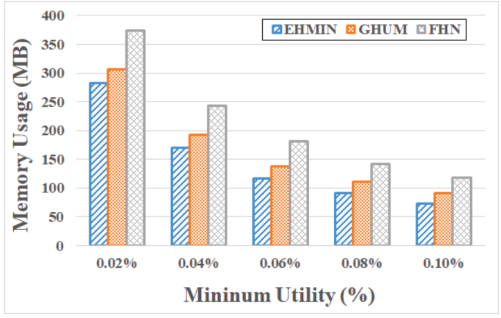


**Fig. 27.** Memory usage tests on the Chainstore and Retail datasets.

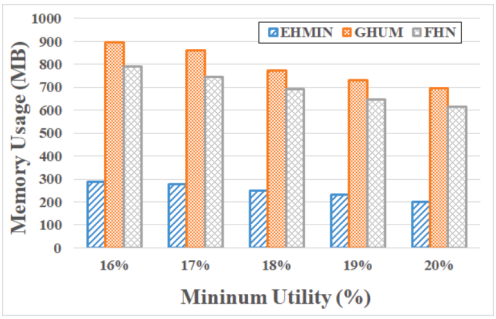


**Fig. 28.** Memory usage tests on and Accidents and Foodmart datasets.

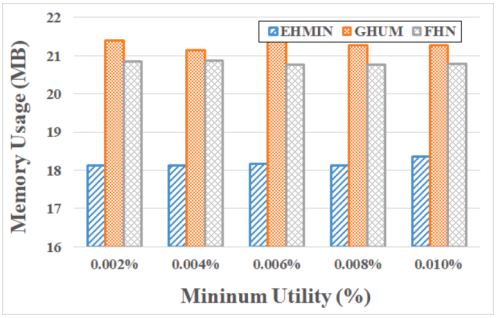


**Fig. 29.** Memory usage tests on and Accidents and Foodmart datasets.

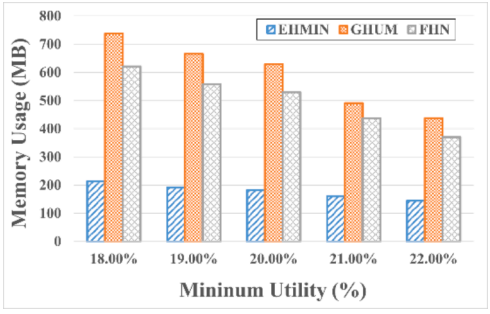


**Fig. 30.** Memory usage tests on and Pumsb and Bms-pos datasets.

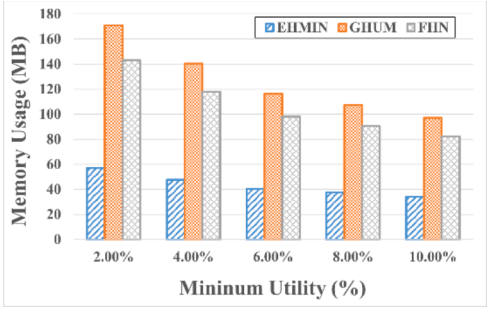


**Fig. 31.** Memory usage tests on and Pumsb and Bms-pos datasets.

complexity analysis prove the completeness of EHMIN.

### 4.3. Analysis of memory usage test

The results of the runtime performances of the algorithms are shown and analyzed in this section. In all the tests, the threshold settings were decided so that the number of result patterns is different on each

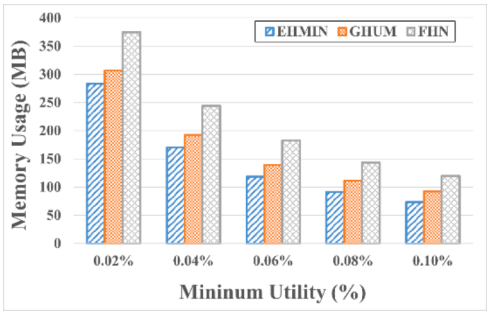


**Fig. 32.** Memory usage tests by changing internal utility.

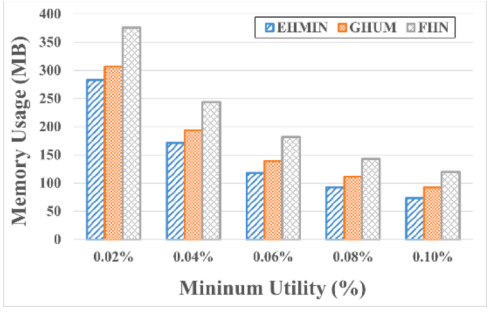


**Fig. 33.** Memory usage tests by changing internal utility.

threshold. Figs. 22–33 are bar graphs that show the results of the memory usage performances on the real datasets. In each dataset, the threshold constantly increases. If the threshold, ***minUtil***, decreases, the memory usage also generally increases. The reason is that the number of patterns to be processed increases. The memory usage depends mainly on the main structures, such as the EHMIN-lists. The EHMIN-lists use vector containers in order to hold the transaction information of patterns, whereas the others usually use map containers for the unnecessary sorting. The structure of a vector container is simpler and more compact than a map container, so vector containers use less memory space than map containers.

Firstly, Fig. 22 shows the memory usage results on the Chess dataset where the threshold increases from 20.0 % to 30.0 %. When the threshold decreases, the memory usage of EHMIN increases slowly, whereas the memory usage of the GHUM and FHN increase quickly. When the threshold is 20.0 %, the EHMIN has a 2.21x and 3.67x better memory usage performance than GHUM and FHN. Next, Fig. 23 represents the memory usage results on the Connect dataset, which is where the threshold increases from 35.00 % to 38.00 %. The memory usage results of the algorithms increase in a similar manner as the threshold decreases. When the threshold is 35.00 %, the EHMIN has a 3.10x and 2.78x better memory usage performance than the GHUM and FHN. Next, Fig. 24 shows memory usage results on the Mushroom dataset, which is where the threshold increases from 2.0 % to 10.0 %. The memory usage results of the algorithms increase in a similar aspect as the threshold decreases. When the threshold is 2.0 %, EHMIN has a 2.18x and 2.05x better memory usage performance than the GHUM and FHN. Next, Fig. 25 indicates memory usage results on Kosarak dataset where the threshold increases from 1.0 % to 3.0 %. The memory usage results of the algorithms increase in the similar shape as the threshold decreases. Where the threshold is 1.0 %, EHMIN has 2.43x and 2.28x better memory usage performance than the GHUM and FHN. Next, Fig. 26 represents memory usage results on the Chainstore dataset where the threshold increases from 0.025 % to 0.125 %. The memory usage results of the algorithms increase in a similar shape as the threshold decreases. Where the threshold is 0.025 %, EHMIN has a 227.8 MB and 421.8 MB better memory usage performance than the GHUM and FHN,

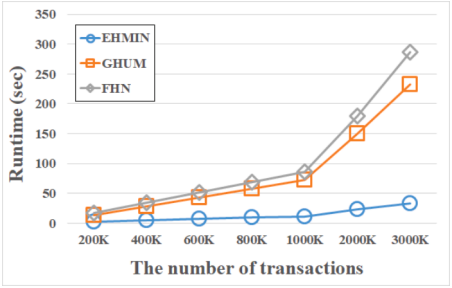


**Fig. 34.** Runtime and Memory usage tests on the T10I4DxK.

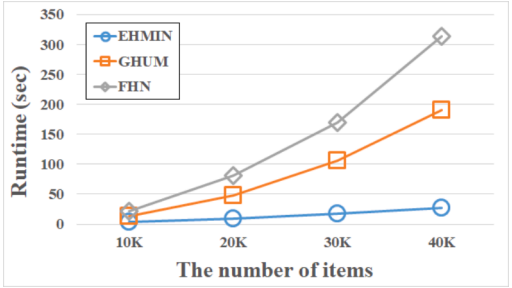


**Fig. 36.** Runtime and Memory usage tests on the TxNx000Lx00.

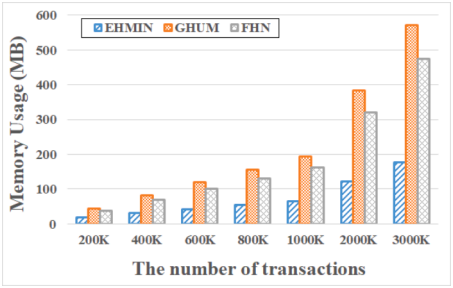


**Fig. 35.** Runtime and Memory usage tests on the T10I4DxK.

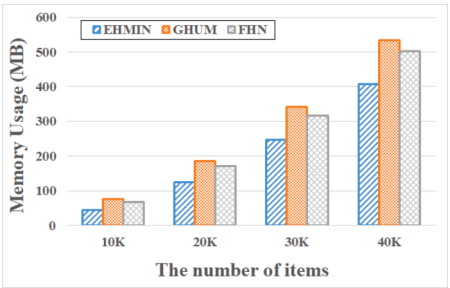


**Fig. 37.** Runtime and Memory usage tests on the TxNx000Lx00.

respectively. Next, Fig. 27 shows memory usage results on the Retail dataset where the threshold increases from 0.02 % to 0.10 %. The memory usage results of the algorithms increase in a similar shape as the threshold decreases. When the threshold is 0.02 %, the EHMIN has a 23.9 MB and 92.2 MB better memory usage performance than GHUM and FHN, respectively. Finally, Fig. 28 represents the memory usage results on the Accidents dataset where the threshold increases from 16 % to 20 %, and Fig. 29 shows the memory usage results on the Foodmart dataset where the threshold increases from 0.002 % to 0.010 %. Fig. 30 and Fig. 31 represents the memory usage in the Pumsb and Bms-pos datasets. In the Pumsb dataset, the thresholds are set from 18 % to 22 %. The Bms-pos dataset is measured in the thresholds range from 2 % to 10 %. Fig. 32 and Fig. 33 are additional memory usage tests on the Retail dataset. The performance of algorithms can vary depending on the internal utility setting, so additional performance evaluations are conducted in various internal utility ranges. On all real datasets and internal utility settings, the EHMIN keeps the best memory usage performance. Moreover, the gaps become larger or are kept as the threshold decreases on all the datasets. This is because the EHMIN has an improved pattern expansion process. The EHMIN efficiently stores transaction information due to the improved pattern extension process. As the threshold decreases, more conditional lists are created, and the difference in the memory usage between the algorithms becomes larger. As a result, the EHMIN has the best performance regardless of the dataset.

### 4.4. Analysis of scalability test

The results of the scalability performances of the algorithms are shown and analyzed in this section. The real datasets have a different number of items and/or a variable number of transactions, but the synthetic datasets have a constant number of items or a constant number of transactions. The first group of items, which is T10I4DxK, has 1000 items and consists of 7 datasets that have 200 K, 400 K, 600 K, 800 K, 1000 K, 2000 K, and 3000 K transactions. In addition, the second group of items, which is TxNx000Lx00, has 100,000 transactions and consists of 4 datasets having 10 K, 20 K, 30 K, and 40 K items. Each threshold setting in the two groups is decided as a constant, so the number of result patterns is varied on the number of transactions or items.

Figs. 34 and 35 illustrate the results of the runtime and memory usage performance on the number of transactions using the group of T10I4DxK datasets. The number of items determines the number of global EHMIN-lists, and the number of transactions affects the average size of the TI-vectors in the EHMIN-lists. Thus, the results show the runtime and memory usage performance on the average size of the TI-vectors. In Fig. 34, all the runtime results increase as the number of transactions increases, because the average size of the TI-vectors that have to be visited in order to construct the conditional EHMIN-lists becomes larger. However, the EHMIN uses the efficient visiting technique by Lemma 1, so the EHMIN is the fastest algorithm of the algorithms in Fig. 34. When the threshold is 0.005 %, and the number of transactions is 1000 K, the EHMIN has a 6.25x and 7.44x better runtime performance than the GHUM and FHN, respectively. In Fig. 35, all the memory usage results also increase as the number of transactions increases. The reason that all the memory usage results increase as the number of transactions increases is that the EHMIN uses vector containers to hold the transactions information of patterns. When the threshold and the number of transaction is 0.005 % and 1000 K, respectively, EHMIN has a 3.01x and 2.52x better memory usage performance than the GHUM and FHN. The gaps of the runtime and memory usage performance become larger as the number of transactions increases. Therefore, it is clear that the EHMIN has the best performance in terms of the transaction scalability in the algorithms.

Figs. 36 and 37 show the results of the runtime and memory usage performance on the number of items using the group of TxNx000Lx00 datasets. The number of items determines the number of global EHMIN-lists, and the number of transactions affects the average size of the TI-vectors in the EHMIN-lists. Thus, the results show the runtime and memory usage performance on the number of global EHMIN-lists. In Fig. 36, all the results for the runtime performance increase as the number of items increases. When the threshold is 0.01 %, and the number of items are 40 K, EHMIN has a 7.07x and × better runtime performance than the GHUM and FHN, respectively. In Fig. 37, all the results of memory usage performance increase as the number of items increases. Where the threshold and the number of items are 0.01 % and 40 K, respectively, the EHMIN has 128.7 MB and 95.6 MB better memory usage performance than GHUM and FHN, respectively. Therefore, it is clear that the EHMIN has the best performance also in terms of

the item scalability in the algorithms.

## 5. Conclusions

In this paper, we suggested an efficient algorithm to perform high-utility pattern mining considering negative unit profits, EHMIN (Efficient High-utility pattern MIning with Negative unit profits). In order to improve the mining performance of the state-of-the-art algorithms in high-utility pattern mining with negative unit profits, we designed a list-based data structure that adopted vector containers to hold transaction information. Moreover, we presented a search technique and a utility calculating technique in order to reduce the construction time of the conditional data structures. In addition, we integrated the exiting pruning strategies. Subsequently, we described how our approach can be implemented in computer languages, and we analyzed the complexity of this algorithm. Finally, the performance evaluation using both real and synthetic databases showed that the suggested approach has a better performance than the existing algorithms in terms of the runtime, memory usage, and scalability. Although EHMIN shows faster performance than the existing algorithms, there are difficulties with the real-time data processing. In our future research, we will improve the algorithm based on the proposed list structure in order to process the database in real time.

## CRediT authorship contribution statement

**Heonho Kim:** . **Taewoong Ryu:** . **Chanhee Lee:** . **Hyeonmo Kim:** . **Eunchul Yoon:** Validation, Conceptualization. **Bay Vo:** Validation, Conceptualization. **Jerry Chun-Wei Lin:** Validation, Conceptualization. **Unil Yun:** Conceptualization, Funding acquisition, Project administration, Methodology, Supervision, Validation, Writing – review & editing.

## 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

The real datasets used in performance evaluations can be obtained from FIMI repository (http://fimi.ua.ac.be/data/)

## Acknowledgements

This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF No. 2021R1A2C1009388).

## References

Ahmed, A. U., Ahmed, C. F., Samiullah, M.d., Adnan, N., & Leung, C. K. (2016). Mining interesting patterns from uncertain databases. *Information Sciences, 354*, 60–85. https://doi.org/10.1016/j.ins.2016.03.007

Ahmed, C. F., Tanbeer, S. K., Jeong, B., & Lee, Y. (2009). Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering, 21*(12), 1708–1721. https://doi.org/10.1109/TKDE.2009.46

Ahmed, C. F., Tanbeer, S. K., Jeong, B., & Lee, Y. (2011). HUC-Prune: An efficient candidate pruning technique to mine high utility patterns. *Applied Intelligence, 34*(2), 181–198. https://doi.org/10.1007/s10489-009-0188-5

Baek, Y., Yun, U., Yoon, E. & Fournier-Viger, P. (2019) Uncertainty-Based Pattern Mining for Maximizing Profit of Manufacturing Plants With List Structure. *IEEE Transactions on Industrial Electronics*, 99, 1-1. 10.1109/TIE.2019.2956387.

Baek, Y., Yun, U., Kim, H., Nam, H., Kim, H., Lin, J. C. W., … Pedrycz, W. (2021). RHUPS: Mining Recent High Utility Patterns with Sliding Window–based Arrival Time Control over Data Streams. *ACM Transactions on Intelligent Systems and Technology (TIST), 12*(2), 1–27. https://doi.org/10.1145/3430767

Cafaro, M., Epicoco, I., & Pulimeno, M. (2019). Mining frequent items in unstructured P2P networks. *Future Generation Computer Systems, 95*, 1–16. https://doi.org/10.1016/j.future.2018.12.030

Chapela-Campa, D., Mucientes, M., & Lama, M. (2019). Mining frequent patterns in process models. *Information Sciences, 472*, 235–257. https://doi.org/10.1016/j.ins.2018.09.011

Chen, Y., Yuan, P., Qiu, M., & Pi, D. (2019). An indoor trajectory frequent pattern mining algorithm based on vague grid sequence. *Expert Systems with Applications, 118*, 614–624. https://doi.org/10.1016/j.eswa.2018.08.053

Chu, C., 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

Deng, N., Chen, X., Li, D., & Xiong, C. (2019). Frequent Patterns Mining in DNA Sequence. *IEEE Access, 7*, 108400–108410. https://doi.org/10.1109/ACCESS.2019.2933044

Dong, X., Gong, Y., & Cao, L. (2018). F-NSP+: A fast negative sequential patterns mining method with self-adaptive data storage. *Pattern Recognition, 84*, 13–27. https://doi.org/10.1016/j.patcog.2018.06.016

Dong, X., Qiu, P., Lu, J., Cao, L., & Xu, T. (2019). Mining Top-k Useful Negative Sequential Patterns via Learning. *IEEE Transactions on Neural Network and Learning Systems, 30*(9), 2764–2778. https://doi.org/10.1109/TNNLS.2018.2886199

Fournier-Viger, P., Wu, C., Zida, S., & Tseng, V. S. (2014). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. *International Symposium on Methodologies for Intelligent Systems, 83–92*. https://doi.org/10.1007/978-3-319-08326-1_9

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

Gan, W., Lin, J.C., Fournier-Viger, P., Chao, H., & Yu, P.S. (2019a). A Survey of Parallel Sequential Pattern Mining. *ACM Transactions on Knowledge Discovery from Data*, 13 (3), 25:1-25:34. 10.1145/3314107.

Gan, W., Lin, J. C., Fournier-Viger, P., Chao, H., & Yu, P. S. (2019b). HUOPM: High Utility Occupancy Pattern Mining. *IEEE Transactions on Cybernetics, 50*(3), 1195–1208. https://doi.org/10.1109/TCYB.2019.2896267

Kim, H., Yun, U., Vo, B., Lin, J. C. W., & Pedrycz, W. (2020). Periodicity-oriented data analytics on time-series data for intelligence system. *IEEE Systems Journal, 15*(4), 4958–4969. https://doi.org/10.1109/JSYST.2020.3022640

Kim, H., Yun, U., Baek, Y., Kim, J., Vo, B., Yoon, E., & Fujita, H. (2021). Efficient list based mining of high average utility patterns with maximum average pruning strategies. *Information Sciences, 543*, 85–105. https://doi.org/10.1016/j.ins.2020.07.043

Kim, J., Yun, U., Yoon, E., Lin, J. C. W., & Fournier-Viger, P. (2020). One scan based high average-utility pattern mining in static and dynamic databases. *Future Generation Computer Systems, 111*, 143–158. https://doi.org/10.1016/j.future.2020.04.027

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. (2018). Efficiently mining high utility itemsets with negative unit profits. *Knowledge-Based Systems, 145*, 1–14. https://doi.org/10.1016/j.knosys.2017.12.035

Kumar, N. S., & Thangamani, M. (2019). Parallel Semi-supervised enhanced fuzzy Co-Clustering (PSEFC) and Rapid Association Rule Mining (RARM) based frequent route mining algorithm for travel sequence recommendation on big social media. *Concurrency and Computation: Practice and Experience, 31*(14). https://doi.org/10.1002/cpe.4837

Lan, G., Hong, T., Huang, J., & Tseng, V. S. (2014). On-shelf utility mining with negative item values. *Expert Systems with Applications, 41*(7), 3450–3459. https://doi.org/10.1016/j.eswa.2013.10.049

Li, H., Huang, H., & Lee, S. (2011). Fast and memory efficient mining of high-utility itemsets from data streams: With and without negative item profits. *Knowledge and Information Systems, 28*(3), 495–522. https://doi.org/10.1007/s10115-010-0330-z

Liu, M. & Qu, J. (2012). Mining high utility itemsets without candidate generation. *21st CIKM*, 55-64. 10.1145/2396761.2396773.

Lan, G., Hong, T., & Tseng, V. S. (2012). An efficient gradual pruning technique for utility mining. *International Journal of Innovative Computing Information and Control, 8*, 5165–5178.

Lee, G., & Yun, U. (2018a). Performance and characteristic analysis of maximal frequent pattern mining methods using additional factors. *Soft Computing, 22*(13), 4267–4273. https://doi.org/10.1007/s00500-017-2820-3

Lee, G., & Yun, U. (2018b). Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases. *Future Generation Computer Systems, 80*, 12–28. https://doi.org/10.1016/j.future.2017.07.035

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

Li, Y., Yeh, J., & Chang, C. (2008). Isolated items discarding strategy for discovering high utility itemsets. *Data & Knowledge Engineering, 64*(1), 198–217. https://doi.org/10.1016/j.datak.2007.06.009

Lin, J. C., 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

Lin, J. C., Yang, L., Fournier-Viger, P., & Hong, T. (2019). Mining of skyline patterns by considering both frequent and utility constraints. *Engineering Applications of Artificial Intelligence, 77*, 229–238. https://doi.org/10.1016/j.engappai.2018.10.010

Liu, Y., Liao, W., & Choudhary, A. (2005). A two-phase algorithm for fast discovery of high utility itemsets. *PAKDD, 689–695*. https://doi.org/10.1007/11430919_79

Nam, H., Yun, U., Yoon, E., & Lin, J. C. W. (2020). Efficient approach for incremental weighted erasable pattern mining with list structure. *Expert Systems with Applications, 143*, Article 113087. https://doi.org/10.1016/j.eswa.2019.113087

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-The Journal of Knowledge. Engineering, 35*(6). https://doi.org/10.1111/exsy.12296

Subramanian, K., & Premalatha, K. (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

Tianrui, Z., Mingqi, W., & Bin, L. (2018). An Efficient Parallel Mining Algorithm Representative Pattern Set of Large-Scale Itemsets in IoT. *IEEE Access, 6*, 79162–79173. https://doi.org/10.1109/ACCESS.2018.2884888

Tseng, V. S., Shie, B., Wu, C., & Yu, P. S. (2012). Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering, 25*(8), 1772–1786. https://doi.org/10.1109/TKDE.2012.59

Tseng, V. S., Wu, C., Shie, B., & Yu, P. S. (2010). UP-Growth: An efficient algorithm for high utility itemset mining. *KDD, 253–262*. https://doi.org/10.1145/1835804.1835839

Pyun, G., & Yun, U. (2014). Mining top-k frequent patterns with combination reducing techniques. *Applied Intelligence, 41*(1), 76–98. https://doi.org/10.1007/s10489-013-0506-9

Yao, H., & Hamilton, H. J. (2006). Mining itemset utilities from transaction databases. *Data & Knowledge Engineering, 59*(3), 603–626. https://doi.org/10.1016/j.datak.2005.10.004

Yun, U., Kim, D., Yoon, E., & Fujita, H. (2018). Damped window based high average utility pattern mining over data streams. *Knowledge-Based Systems, 144*, 188–205. https://doi.org/10.1016/j.knosys.2017.12.029

Yun, U., Lee, G., & Yoon, E. (2019a). Advanced approach of sliding window based erasable pattern mining with list structure of industrial fields. *Information Sciences, 494*, 37–59. https://doi.org/10.1016/j.ins.2019.04.050

Yun, U., Nam, H., Lee, G., & Yoon, E. (2019b). 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

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), 1–17. https://doi.org/10.1142/S0218001417500355

Xu, T., Li, T., & Dong, X. (2018). Efficient High Utility Negative Sequential Patterns Mining in Smart Campus. *IEEE Access, 6*, 23839–23847. https://doi.org/10.1109/ACCESS.2018.2827167

Zhang, L., Yang, S., Wu, X., Cheng, F., Xie, Y., & Lin, Z. (2019). An indexed set representation based multi-objective evolutionary approach for mining diversified top-k high utility patterns. *Engineering Applications of Artificial Intelligence, 77*, 9–20. https://doi.org/10.1016/j.engappai.2018.09.009

Zhang, S., Zhang, Y., Yin, L., Yuan, T., Wu, Z., & Luo, H. (2019). Mining Frequent Items Over the Distributed Hierarchical Continuous Weighted Data Streams in Internet of Things. *IEEE Access, 7*, 74890–74898. https://doi.org/10.1109/ACCESS.2019.2911573

Zhu, X., Liu, Y., Li, Q., Zhang, Y., & Wen, C. (2018). Mining Effective Patterns of Chinese Medicinal Formulae Using Top-k Weighted Association Rules for the Internet of Medical Things. *IEEE Access, 6*, 57840–57855. https://doi.org/10.1109/ACCESS.2018.2873677