IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 March 2023, accepted 20 April 2023, date of publication 1 May 2023, date of current version 5 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3271864

RESEARCH ARTICLE

Efficient Method for Mining High Utility Occupancy Patterns Based on Indexed List Structure

HYEONMO KIM[1], TAEWOONG RYU[1], CHANHEE LEE[1], SINYOUNG KIM[1], BAY VO[2], (Member, IEEE), JERRY CHUN-WEI LIN[3], (Senior Member, IEEE), AND UNIL YUN[1]

[1]Department of Computer Engineering, Sejong University, Seoul 05006, South Korea
[2]Faculty of Information Technology, Ho Chi Minh University of Technology (HUTECH), Ho Chi Minh City 700000, Vietnam
[3]Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5063 Bergen, Norway

Corresponding author: Unil Yun (yunei@sejong.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology under Grant 2021R1A2C1009388.

ABSTRACT High utility pattern mining has been proposed to improve the traditional support-based pattern mining methods that process binary databases. High utility patterns are discovered by effectively considering the quantity and importance of items. Recently, high utility occupancy pattern mining studies have been conducted to extract high-quality patterns by utilizing both the occupancy utility and frequency measure. Although the previous approaches provide worthy information in terms of utility occupancy, they require time-consuming tasks because of numerous comparison operations in exploring entries in global data structures. This results in significant performance degradation when the database is large, or a pre-defined threshold is low. An indexed list structure improves the inefficiency of the list-based approach by structurally connecting each tuple. In this paper, we propose an efficient high utility occupancy mining approach based on novel indexed list-based structures. The two newly designed data structures maintain index information on items or patterns and facilitate rapid pattern extension. Our approach improves the cost of generating long patterns of list-based ones by reducing a large number of comparison overheads. In addition, we devise novel constructing and mining methods that are suitable for the proposed data structures and utility occupancy functions. To narrow the wide search space, efficient pruning techniques apply to the designed methods. Thorough performance experiments using real and synthetic datasets show that our method is more efficient than state-of-the-art methods in environments where given thresholds change.

INDEX TERMS Data mining, high-utility occupancy pattern, indexed list structure, pattern mining.

I. INTRODUCTION

Due to the development of information technology, many applications generate and handle a lot of data. As a huge amount of data accumulates, data analysis techniques that discover meaningful knowledge, such as relations and association rules, from large data have been actively studied. Data mining is one of the most promising data analysis techniques. Pattern mining, which is a sub-area of data mining, discovers a valuable set of items, called a pattern or an itemset, from a given database. Frequent pattern mining [1], [2], [3] is a fundamental subject in the field of pattern mining. It mines a frequent pattern whose support, which means the number of times a pattern occurs in the database, is greater than or equal to a user-given threshold. Since frequent patterns can be used to help the decision maker, studies on more efficient frequent pattern mining have been conducted in various fields. However, in the real world, there are applications that are interested in a pattern that not only appears frequently but also plays an important role in transactions containing the pattern. In order to meet the requirements in the real world, high occupancy pattern mining [4], [5], [6] was proposed. High occupancy pattern mining discovers more representative and qualified patterns by considering the occupancy that

The associate editor coordinating the review of this manuscript and approving it for publication was Hong-Mei Zhang.

represents a proportion of a pattern to the transaction. However, both frequent pattern mining and occupancy pattern mining have limitation in that they ignore the implicit factors, such as weight, profit, importance, and quantity.

High utility pattern mining [7], [8], [9] handles the quantitative transaction database where each item has a quantity and relative profit, and extracts a pattern whose utility, which is the product of a quantity and profit, is not less than a user-specified threshold. Compared to the frequent pattern, the high utility pattern has a high profit by processing more factors, including the quantities and profits of items. Therefore, high utility pattern mining is utilized in various applications [10]. Moreover, expanded research on high utility pattern mining has been performed by combining the utility concept with approaches based on other notions, such as sequential pattern mining [11], maximal pattern mining [12], fuzzy pattern mining [13], and non-redundant pattern mining [14]. However, high utility pattern mining cannot fully reflect the frequency of the pattern. This is because a pattern that occurs once or rarely can be extracted as a high utility pattern. Providing these patterns as result patterns to the decision maker can lead to poor decisions. In addition, as a pattern appears frequently, the utility of the pattern increases. Hence, although a pattern does not have any significant influence on transactions, the pattern can be mined as a high utility pattern and misleads users if the pattern occurs frequently in the database.

To overcome the above problems, OCEAN [15], the high utility occupancy pattern mining method, was first proposed. High utility occupancy pattern mining considers not only the quantity and relative profit but also the ratio of the utility of the pattern in each transaction. Moreover, in high utility occupancy pattern mining, support and utility occupancy are used as measures. The two measures guarantee that the extracted result patterns are frequent and highly qualified. OCEAN adopts a list-based vertical data structure, named as a utility list [16], for storing utility information of a pattern. The list structure is accompanied by join-operations, which involve vertical search to find identical transactions, consume significant costs. HUOPM [17] tried to improve the performance by utilizing UO-list, which stores the utility occupancy information, and the frequency-utility table (FU-Table). However, since UO-list is also a list-based vertical data structure, it still has the problem of computational costs for join-operations. Additionally, in order to address the performance reduction facing OCEAN, HUOPM adopts several pruning techniques that actively utilize the properties of support and utility occupancy. The previous utility occupancy pattern mining methods offer interesting results to the users, but they still have an excessive number of comparison operations in their mining process. In addition, the consumption to calculate the upper bound causes performance degradation. As the size of the database increases, the list-based structures contain many entries corresponding to the transactions. This may result in considerable comparison operations, thereby it is inefficient for processing large-scale databases. As a result, this prevents users from making quick decisions. Meanwhile, an indexed list structure improves the problem by keeping index information of items or patterns. However, the current indexed lists are not capable of solving the high utility occupancy pattern mining problem.

Motivated and inspired by these challenges, we propose an efficient high utility occupancy pattern mining approach, named HUOMIL (High Utility Occupancy pattern Mining with Indexed List), using an indexed list-based data structure. HUOMIL reduces the computational costs in the mining process by employing the indexed list structure and an additional pruning strategy to narrow search space. Furthermore, the methods capable of effectively addressing utility occupancy functions are newly devised and included in the proposed approach. To the best of our knowledge, this is the first work that utilizes the indexed list structure in the high utility occupancy pattern mining area. The main contributions of this paper are as follows:

- We propose an efficient high utility occupancy pattern mining approach utilizing the indexed list. The proposed approach includes effective and efficient methods that rely on an indexed list-based framework. Based on these methods, we develop an algorithm to discover high-quality patterns with efficiency.
- Two novel indexed list-based data structures, GUO-IL and CUO-IL, are devised to maintain the essential information required to mine high utility occupancy patterns. They include quadruple entries that contain index information of items or patterns.
- The mining method designed based on the proposed data structures tracks all entries by accessing only relevant entries. This leads to a reduction of the number of comparison operations while creating long patterns.
- We prove that the proposed algorithm outperforms state-of-the-art high utility occupancy pattern mining approaches in terms of runtime, peak memory usage, and scalability using diverse real and synthetic datasets.

The rest of this paper is organized as follows. Section II introduces the studies related to the suggested approach. In Section III, preliminaries are given first. Then, the overall process of the proposed method, the proposed indexed list structure, and the mining process are described in detail. Section IV analyzes performance evaluation results conducted on various datasets against the latest algorithms. In Section V, limitations and future directions are discussed. Finally, the conclusions are presented in Section VI.

II. RELATED WORK

A. UTILITY-DRIVEN PATTERN MINING

High utility pattern mining finds meaningful patterns from the quantitative database consisting of quantity and profit. Since the results provide users with set of items that yields high profits. Therefore, high utility pattern mining is used to analyze retail [18], IoT systems [19], [20], operation patterns [21], and financial crises [22] where the quantity and profit of items are considered important. Starting with a traditional approach, various methods that leverage different strategies for high efficiency were proposed. The two-phase algorithm [23] is a fundamental algorithm that laid the foundation for this field. In addition, it suggested TWU-model satisfying anti-monotone property to prune unpromising patterns in advance. Nevertheless, it has performance limitations due to the huge number of candidate patterns and database scans. Some tree-based algorithms effectively tackle the problem by reducing the number of scans. IHUP [24] utilizes a pattern growth method rather than the level-wise manner. Moreover, the number of scans is considerably less than that of two-phase, thereby they show better performance in experiments. UP-Growth and UP-Growth+ [25] perform the mining process with tight upper bounds rather than TWU values. Meanwhile, when they generate a vast number of candidate patterns, an additional scan for determining their validity causes performance degradation. The use of a list data structure can solve this problem without generating candidates because it stores the utility of items in the database. HUI-Miner [16] used a vertical list structure, called a utility list, and crucial definitions, including remaining utility or sorting order. The method can report results with fewer scans. On top of that, an upper bound tighter than those of the above algorithms improves pruning efficiency. Since the introduction of HUI-Miner, several algorithms relying on a list structure have been developed and representative of them are FHM [26], HUP-Miner [27], and HMiner [28]. Despite these efforts, as the number of transactions to which items belong increases, the volume of each list increasingly grows. In this aspect, it is costly to traverse all entries in lists during the pattern extension. The indexed list structure [29], [30], [31] improves the previous list structure by storing the path to the next items. They efficiently explore patterns through the index of the next item and improve the runtime and memory usage of high utility pattern mining.

High utility pattern mining has been evolved in a variety of ways to reflect real world factor or scenarios. High average utility pattern mining [32], [33], [34] considers the utility and length of the pattern simultaneously. The pre-large concept [35], [36] finds large and pre-large patterns to minimize database scans. RUPM [37] extracts reliable high utility patterns by adapting the concept of reliability. Closed high utility pattern mining [38], [39] generates representative patterns to simplify extracted high utility patterns. MCH-Miner [40] efficiently extracts high utility patterns by applying a parallel multi-core architecture. Correlated high utility pattern mining [41], [42] extracts a high utility pattern considering the correlation of items. RHUPS [43] adopts a sliding window and list to find recent high utility patterns. EHMIN [44] applies a list structure to efficiently find high utility patterns with negative profits. DHUPL [45] finds damped utility patterns that consider the order in which transactions occur. DHUP-Miner [46] extracts affinity utility patterns through a list structure. MLHMiner [47] adopts the HMiner structure to find multi-level high utility patterns. These approaches provided more meaningful insight with practical analysis than traditional ones. They, however, sometimes fail to meet the needs of applications, such as recommendation-based system. The measure, called occupancy, effectively considers the proportion of importance that the pattern occupies, which also applies to the approach proposed in this paper. On top of that, despite the efficiency of the indexed list structure in the high utility pattern mining, there is no research that considers occupancy measure and utilizes the indexed list structure.

B. OCCUPANCY-DRIVEN PATTERN MINING

The challenge of pattern mining [48] involves finding a more interesting pattern and speeding up the mining process. High occupancy pattern mining is the practical research more useful than the conventional ones that rely on frequent and utility concepts. Occupancy pattern mining considers the ratio of a pattern in a transaction, which is regarded as occupancy. Even if a pattern appears frequently, it cannot be a result pattern if the pattern has a low occupancy for its transactions. Thus, by this measure, occupancy pattern mining is used to discover the intent or purpose of consumers from transactions. DOFRA [6] finds patterns by considering the support and average occupancy from transactions or sequences. HEP [5] integrated two thresholds, support and occupancy. They propose an accumulated occupancy that includes the properties of support instead of the average occupancy. HOIMTO [49] suggests a high occupancy pattern mining technique that considers transaction occupancy as well as pattern occupancy. HOMI [50] extracts high occupancy patterns from an incremental database using the list structure. CFBPP [51] proposes tire-based occupancy pattern mining to discover licensed empty spectrum patterns. QFWO algorithm [52] is a list-based algorithm. The authors designed a new research problem with the concept of weight and occupancy and defined a new measure called weight occupancy. It could solve the new problem of high-quality patterns by utilizing using a weight-list structure. Combining the occupancy measure with frequency or weight has led to practical information. On the other hand, research on utility-based occupancy has also been steadily conducted.

OCEAN [15] proposed occupancy pattern mining regarding the utility of patterns. It finds items with high utility occupancy from the quantity database. Therefore, utility occupancy pattern mining is useful for identifying the purpose or habit of consumers from a database where the utility of an item is important. HUOPM [17] is proposed to improve the performance of OCEAN. Mining performance is improved through new data structures, which are the utility occupancy list and frequent occupancy table. In addition, the pruning efficiency is increased by changing the sorting order of the list structure. Next, utility occupancy pattern mining has various application fields. SHUO-FI [53] finds utility occupancy patterns considering distance constraints. UHUOPM [54] and HUOMI [55] adopt the list structure and extract utility occupancy patterns from uncertain situations and incremental environments, respectively. pnHUO [56] is an algorithm of the research that defined a new problem by integrating utility occupancy and negative utility concepts. It incorporated two list-based structures inspired by HUOPM. The above list-based methods can derive the result patterns more high-quality than that of frequency-based methods. However, none of the existing utility occupancy pattern mining methods use the superior list structure, indexed list. Therefore, we propose an indexed list-based utility occupancy pattern mining method to improve the performance of the previous methods.

III. PROPOSED APPROACH FOR MINING HIGH UTILITY OCCUPANCY PATTERNS WITH INDEXED LIST

We propose an efficient approach that includes an improved indexed list-based data structure and methods in this paper. The novel data structure enables HUOMIL to realize the reduction of comparison operations during the mining process. This section consists of several subsections detailing the fundamental definitions, the proposed data structure, and how HUOMIL mines the results. In the last part of this section, the operation of HUOMIL is precisely analyzed with a description of the algorithms.

A. PRELIMINARIES

The preliminaries, which are closely related to high utility occupancy pattern mining, are covered comprehensively in this subsection. We specify the rudimentary terms and definitions in this section by referring to the prior studies [15], [17] on high utility occupancy pattern mining. A quantitative database contains multiple transactions, and it is denoted as $QDB = \{T_1, T_2, \ldots, T_m\}$. Each transaction is composed of items and their quantities. When a set of all items in QDB is represented by I and an item $\{i\}$ ($\{i\} \in I$) is in transaction T($T \in QDB$), the quantity of $\{i\}$ is expressed as $q(\{i\}, T)$. Values in an external utility table mean the external utilities of items (e.g., profit, importance, and weight). The external utility of $\{i\}$ is denoted as $p(\{i\})$. The utility of $\{i\}$ in T is calculated by multiplying $q(\{i\}, T)$ and $p(\{i\})$, and it is denoted as $u(\{i\}, T)$. A pattern P is a set of several items. This means that $P \subseteq I$. P is called k-itemset when the number of items in P is k. For example, the quantitative database in Fig. 1 has 7 unique items and 10 transactions. For an item $\{A\}$, $q(\{A\}, T_1)$ is 2, and $p(\{A\})$ is 3. Therefore, $u(\{A\}, T_1)$ becomes 6. In addition, a pattern $\{A, B, C\}$ is called 3-itemset because the number of items in the pattern is 3.

Definition 1: The utility of a pattern X in a transaction T is denoted as $u(X, T)$, and $u(X)$ refers to the total sum of the utilities of X in each transaction T_k in QDB. These two values are formally defined as follows:

$$u(X, T) = \sum u(i_x, T), i_x \in \hat{o}X \cap X \subseteq T \tag{1}$$

Transaction	A set of items (*name*, *quantity*)	TU
T_1	$(A,2)\ (B,3)\ (C,1)$	13
T_2	$(D,2)\ (F,2)$	20
T_3	$(B,4)\ (G,2)$	14
T_4	$(A,3)\ (C,4)\ (E,2)\ (F,1)$	21
T_5	$(B,2)\ (D,1)\ (E,1)$	12
T_6	$(B,4)\ (C,1)\ (D,1)$	15
T_7	$(F,1)\ (G,2)$	10
T_8	$(A,3)\ (B,1)\ (C,5)$	16
T_9	$(F,1)\ (G,4)$	16
T_{10}	$(B,2)\ (C,5)\ (D,2)$	21

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
EU	3	2	1	6	2	4	3

FIGURE 1. Example quantitative database and external utility table.

$$u(X) = \sum u(X, T_k), X \subseteq T_k \cap T_k \in \hat{o}QDB \tag{2}$$

Definition 2: The transaction utility of a transaction T in QDB is notated by $tu(T)$, which stands for the total sum of the utilities of the items in T. $tu(T)$ is obtained by the following formula:

$$tu(T) = \sum u(i_x, T), i_x \in \hat{o}T \tag{3}$$

$$(X) = |T^X| = |\{T | X \subseteq T \cap T \in \hat{o}QDB\}| \tag{4}$$

Definition 3: In frequent pattern mining [2], [3], the support of a pattern X refers to the number of transactions where X appears in QDB, and the notation is $sup(X)$. When a set of transactions containing X is T^X, $sup(X)$ is the cardinality of T^X. Thus, $sup(X)$ is calculated as below:

Definition 4: The utility occupancy of a pattern X in a transaction T indicates the ratio of $u(X, T)$ to $tu(T)$, which is denoted as $uo(X, T)$. In addition, the utility occupancy of X in QDB, expressed as $uo(X)$, is the average of $uo(X, T)$ calculated per transaction. The following two formulas are used to obtain $uo(X, T)$ and $uo(X)$.

$$uo(X, T) = u(X, T)/tu(T)(5) \tag{5}$$

$$uo(X) = \sum uo(X, T_k)/sup(X), T_k \in \hat{o}QDB \tag{6}$$

For example, let us take an example database in Fig. 1 and consider the calculations of $u(\{A, C\})$, $uo(\{A, C\}, T_1)$, and $uo(\{A, C\})$. A pattern $\{A, C\}$ occurs in transactions T_1, T_4, and T_8, so $u(\{A, C\}) = u(\{A, C\}, T_1) + u(\{A, C\}, T_4) + u(\{A, C\}, T_8) = 7+13+14=34$. $uo(\{A, C\}, T_1)$ is calculated as $7/13$, which is approximately 0.5384. Similarly, $uo(\{A, C\}, T_4)$ and $uo(\{A, C\}, T_8)$ are roughly 0.6189 and 0.875, respectively. Based on these uo values, $uo(\{A, C\})$ is (0.5384+0.6189+0.875)/3, which is about 0.6774.

Definition 5: When two threshold ratios are specified by the users, γ ($0 < \gamma \leq 1$) and δ ($0 < \delta \leq 1$), the minimum support threshold and minimum utility occupancy threshold are denoted by $minSup(QDB, \gamma)$ and $minUtilOcc(\delta)$, and they are obtained as follows:

$$minSup(QDB, \gamma) = |QDB| \times \gamma \tag{7}$$

$$minUtilOcc(\delta) = \delta \tag{8}$$

Definition 6: In high utility occupancy pattern mining [15], [17], a pattern X in QDB is called a high utility occupancy pattern if both $sup(X) \geq minSup(QDB, \gamma)$ and $uo(X) \geq minUtilOcc(\delta)$ are satisfied. The high utility occupancy pattern implies that the pattern has a high occupancy with respect to the utility as well as a high frequency in QDB.

For example, consider three patterns, $\{A, C\}$, $\{A, F\}$, and $\{C\}$, which can be generated in the example database. When $minSup(QDB, \gamma)$ and $minUtilOcc(\delta)$ are 3 and 0.3, $\{A, C\}$ becomes a high utility occupancy pattern because $sup(\{A, C\}) = 3$ and $uo(\{A, C\}) = 0.6775$. Meanwhile, $uo(\{A, F\})$ is 0.6190, but $sup(\{A, F\})$ is less than 3. For the pattern $\{C\}$, $sup(\{C\}) = 4$ and $uo(\{C\}) = 0.1769$, which is less than 0.3. Since the patterns $\{A, F\}$ and $\{C\}$ satisfy only one constraint, they are not a high utility occupancy pattern.

Definition 7: Total order, denoted as $\prec$, refers to a specific order in which an algorithm processes items in the database. In the framework of the proposed approach, the total order on a set of items, $I = \{i | sup(i) \geq minSup(QDB, \gamma)\}$, is assumed in support-ascending order of items in I.

The previously designed high utility occupancy pattern mining approaches [15], [17] used the total order in TWU-ascending order and support-ascending order, respectively, for efficient mining. In the indexed list-based framework, the total order follows the support-ascending order.

Definition 8: Let a revised transaction be T. The revised transaction is a transaction that has items whose support is not less than $minSup(QDB, \gamma)$, and the items are sorted in the total order $\prec$. In utility occupancy concept [17], the remaining utility occupancy of a pattern X ($X \subseteq T$) is denoted as $ruo(X, T)$. Furthermore, the remaining utility occupancy of X throughout QDB is notated as $ruo(X)$. When the last item of X is represented by i_x, let i_y be an item located after i_x. Then, $ruo(X, T)$ and $ruo(X)$ are calculated as follows:

$$ruo(X, T) = \sum uo(i_y, T), i_x \prec i_y \cap i_y \in \hat{o}T \tag{9}$$

$$ruo(X) = \sum ruo(X, T_k)/sup(X), X \subseteq T_k \cap T_k \in \hat{o}QDB \tag{10}$$

Definition 9: Suppose that there is a pattern X. The upper bound utility occupancy of X is expressed as $ubuo(X)$. Let $topK$ be the largest top K elements in a set. For each transaction T_k including X, $ubuo(X)$ is the average of the top $\lceil minSup(QDB, \gamma)\rceil$ elements in a set of the sum of $uo(X, T_k)$ and $ruo(X, T_k)$. It is formulated as follows:

$$ubuo(X) = \sum^{\mathrm{Top}\lceil \mathrm{minSup(QDB},\gamma)\rceil} \{uo(X, T_k) + ruo(X, T_k)\} \\ / \lceil minSup(QDB, \gamma)\rceil, X \subseteq T_k \cap T_k \in QDB \tag{11}$$

When a pattern Y ($X \subseteq Y$) is a super pattern of X, $ubuo(X)$ is used for estimating the uo value of Y. No super pattern of X has a utility occupancy greater than $ubuo(X)$. Therefore, if $ubuo(X)$ is less than $minUtilOcc(\delta)$, any pattern Y does not become the high utility occupancy pattern [17]. Moreover, high utility occupancy pattern mining algorithms insert the sum of uo and ruo for each T_k into a vector, and the elements in the vector are sorted in descending order of their values.

For example, given that the total order is alphabetical order and $minSup(QDB, \gamma)$ is 3, $ruo(\{B\})$ is the average of $ruo(\{B\}, T_1)$, $ruo(\{B\}, T_3)$, $ruo(\{B\}, T_5)$, $ruo(\{B\}, T_6)$, $ruo(\{B\}, T_8)$, and $ruo(\{B\}, T_{10})$. The set of items appearing after $\{B\}$ in T_1 is $\{C\}$, so $ruo(\{B\}, T_1) = 1/130.0769$. The remaining ruo values are obtained in the same manner. As a result, $ruo(\{B\})$=(0.0769 + 0.4285 + 0.6666 + 0.4666 + 0.3125 + 0.8095) / 6 0.4601. Next, in order to compute $ubuo(\{B\})$, a vector (Vec) includes the sum of $uo(\{B\}, T_k)$ and $ruo(\{B\}, T_k)$, where T_k contains $\{B\}$, in descending order. Thus, Vec= $\{1, 1, 1, 1, 0.5384, 0.4375\}$. The top three elements are 1,1, and 1, thus $ubuo(\{B\})$=(1 + 1 + 1) / 3 = 1.

B. OVERALL PROCESS OF THE PROPOSED APPROACH WITH INDEXED LIST

Fig. 2 shows the overall operational flow of the proposed approach, HUOMIL, to extract high utility occupancy patterns with efficient indexed list-based structures. HUOMIL carries out three major processes: 1) A database scan to determine essential information, 2) A database scan to construct efficient data structures, and 3) Combinations of patterns to extract a set of high utility occupancy patterns. First, the algorithm scans a quantitative database to investigate the support of items and determines a total order. Then, the database is scanned again to generate the proposed indexed list-based data structures for each item. The constructed indexed lists organize a global set. During the construction, the utility occupancy and remaining utility occupancy of the items are calculated. The indexed lists related to each other are mutually connected by index information. An indexed list-based structure, named GUO-IL, manages a series of quadruple entries that store fundamental information. After that, HUOMIL performs the mining process with the global set. This process adopts a prefix-based pattern extension manner, which first selects a prefix pattern and then recursively extends the prefix in a DFS manner. HUOMIL combines k-itemsets to completely explore all patterns without omission and generates ($k + 1$)-itemsets that conditionally have the selected prefix in common. During this process, indexed list-based conditional lists, called CUO-IL, are created for the next mining step. After that, a new prefix is chosen from the ($k + 1$)-itemsets based on the DFS method. In this recursion, several pruning strategies are deployed in order to efficiently reduce unnecessary exploration.

C. CONSTRUCTION OF THE PROPOSED INDEXED LIST-BASED DATA STRUCTURES

1) THE PROPOSED INDEXED LIST-BASED DATA STRUCTURES

In this section, we introduce two types of new data structures called global utility occupancy indexed list (GUO-IL) and conditional utility occupancy indexed list (CUO-IL). To quickly expand patterns into long patterns, GUO-IL and CUO-IL employ index information. Fig. 3 expresses the structural form of GUO-IL and CUO-IL.

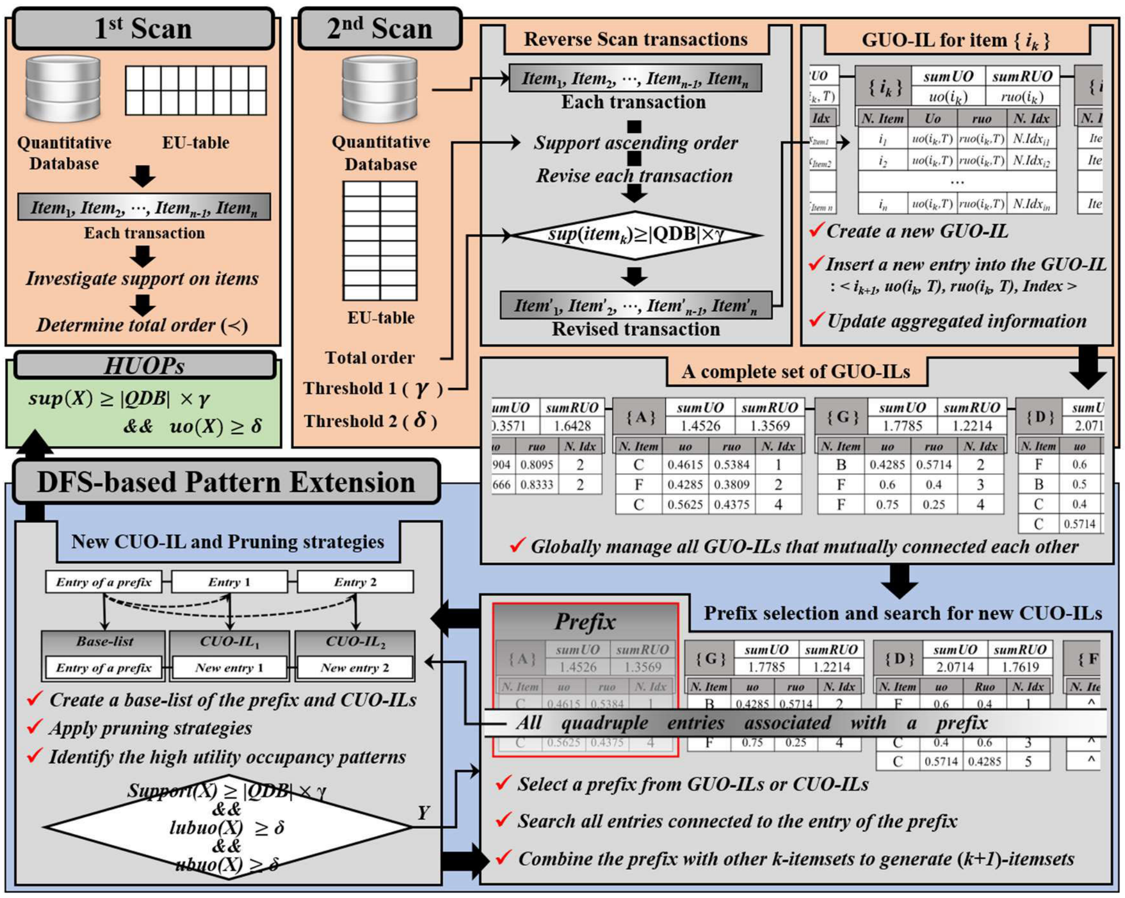
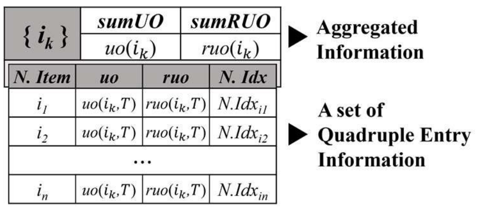
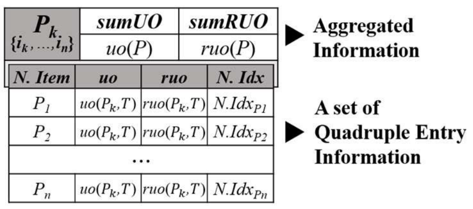

1st Scan
Quantitative Database
EU-table
Item1, Item2, ···, Itemn-1, Itemn
Each transaction
Investigate support on items
Determine total order (≺)
2nd Scan
Quantitative Database
EU-table
Total order
Threshold 1 (γ)
Threshold 2 (δ)
Reverse Scan transactions
Each transaction
Support ascending order
Revise each transaction
sup(itemk)≥|QDB|×γ
Revised transaction
GUO-IL for item { ik }
Create a new GUO-IL
Insert a new entry into the GUO-IL : < ik+1, uo(ik, T), ruo(ik, T), Index >
Update aggregated information
HUOPs
sup(X) ≥ |QDB| × γ && uo(X) ≥ δ
A complete set of GUO-ILs
Globally manage all GUO-ILs that mutually connected each other
DFS-based Pattern Extension
New CUO-IL and Pruning strategies
Entry of a prefix
Entry 1
Entry 2
Base-list
CUO-IL1
CUO-IL2
New entry 1
New entry 2
Create a base-list of the prefix and CUO-ILs
Apply pruning strategies
Identify the high utility occupancy patterns
Support(X) ≥ |QDB| × γ && lubuo(X) ≥ δ && ubuo(X) ≥ δ
Y
Prefix
Prefix selection and search for new CUO-ILs
All quadruple entries associated with a prefix
Select a prefix from GUO-ILs or CUO-ILs
Search all entries connected to the entry of the prefix
Combine the prefix with other k-itemsets to generate (k+1)-itemsets

FIGURE 2. Overall process of HUOMIL.



(a) Global Utility Occupancy Indexed List (GUO-IL)



(b) Conditional Utility Occupancy Indexed List (CUO-IL)

FIGURE 3. The structural form of the proposed indexed lists, GUO-IL and CUO-IL.

As shown in Fig. 3 (a), GUO-IL includes two kinds of information, called aggregated information and a set of entries. The set of entries manages *uo*, *ruo*, next item (*N.Item*), and next index (*N.Idx*) for an item $\{i_k\}$. That is, each entry is formed like $< i_{k+1}, uo(i_k, T), ruo(i_k, T), N.Idx_{i(k+1)} >$, where $\{i_{k+1}\}$ is an item after $\{i_k\}$ in a revised transaction T. The sum of *uo* (*sumUO*) and the sum of *ruo* (*sumRUO*) are independently managed in aggregated information. They are used in the mining phase to determine the validity of patterns and to prune useless patterns. The design of CUO-IL, shown in Fig. 3 (b), is similar to that of GUO-IL. However, GUO-IL stores only information about an item, while CUO-IL contains information about a k-itemset ($k \geq 2$). Therefore, it requires dissimilar methods different from that of GUO-IL to calculate utility occupancy and set index information. It will be explained later in the section describing the mining process. The proposed data structures are clearly distinguished from the data structure of OCEAN [15] which is derived from the utility list [16]. In addition, ours differs from the UO-List and FU-table proposed by HUOPM [17]. The previous structures manage the information that an item has in a particular transaction as a triple entry (i.e., <*tid*, *uo*, *ruo*>), whereas GUO-IL and CUO- IL have a quadruple entry which includes the next item and next index information.

2) CONSTRUCTION OF THE GLOBAL UTILITY OCCUPANCY INDEXED LIST STRUCTURE

HUOMIL scans a quantitative database twice to organize a complete set of GUO-ILs. In this section, we describe each database scan process with detailed examples. The support of each item is investigated by scanning the database only once. After that, a total order is determined to maximize the mining efficiency by sorting the supports in ascending order. Note that the total order contains only items whose support is no less than the minimum support threshold. The reason is that a pattern generated from an item with support less than *minSup*(QDB,γ), never becomes a high utility occupancy pattern due to the anti-monotone property [1]. It enables HUOMIL to create GUO-ILs for only promising items that satisfy the minimum support constraint. In the subsequent scan process, each revised transaction, which eliminates unpromising items and arranges the remaining items by the total order, is scanned in reverse order, and GUO-ILs are generated at the same time. Whenever the item is visited, the next item, next index, utility occupancy, and remaining utility occupancy are stored in a new quadruple entry.

Definition 10: A quadruple entry that constitutes a set of entries in GUO-IL contains two significant elements. For an item $\{i_k\}$, the next item of $\{i_k\}$ represents the item $\{i_{k+1}\}$ that is located immediately after $\{i_k\}$ in a revised transaction, $\{\ldots,\ i_k,\ i_{k+1}\ ,\ldots\ \}$. The next index of $\{i_k\}$ refers to the current number of entries in GUO-IL of $\{i_{k+1}\}$. If $\{i_k\}$ is the last item in the transaction, the next item and next index of $\{i_k\}$ have NULL values and are marked with " ".

If an item $\{i_k\}$ in T is processed for the first time, a new GUO-IL for $\{i_k\}$ is newly created and inserted into a set of GUO-ILs. Then, a new quadruple entry stores information about $\{i_k\}$ as follows: $<\ i_{k+1},\ uo(i_k,\ T),\ ruo(i_k,\ T),\ N.Idx_{i(k+1)}\ >$, where $N.Idx_{i(k+1)}$ refers to the location of the entry for $\{i_{k+1}\}$. We can simply calculate $ruo(i_k,\ T)$ and $N.Idx_{i(k+1)}$ because the revised transaction is scanned in reverse order. If GUO-IL of $\{i_k\}$ already exists, an entry is just added there. During the insertion of the entries, *sumUO* and *sumRUO* are updated. After the construction of indexed lists is finished, we finally obtain the aggregated information. The following two running examples describe the processing of HUOMIL in the database scans, respectively.

Items in each transaction are arranged according to $A \prec G \prec D \prec F \prec C \prec B$, where $\{E\}$ is removed because $sup(\{E\})<minSup(QDB,\ \gamma)$. Concretely, a transaction T_1, $\{A,\ B,\ C\}$, is revised to $\{A,\ C,\ B\}$. A revised database composed of revised transactions is shown in Fig. 4. The item $\{B\}$ in T_1 is processed first. There is no GUO-IL($\{B\}$), so it is newly generated. The utility occupancy and remaining utility occupancy of $\{B\}$ in T_1 are obtained as follows: $uo(\{B\},\ T_1) = 6\ /\ 13$ Ö 0.4615 and $ruo(\{B\},\ T_1) = 0$. In addition, *N.Item* and *N.Idx* of $\{B\}$ in T_1 become NULL(^) because $\{B\}$ is the last item in the revised transaction T_1. Consequently, the quadruple entry, $<\ \wedge$,0.4615, 0, $\wedge\ >$, is added to a set of entries of GUO-IL($\{B\}$). After that, *sumUO* and *sumRUO* of GUO-IL($\{B\}$) accumulate 0.4615 and 0, respectively. Next, GUO-IL($\{C\}$) is then created, and $uo(\{C\},\ T_1)$ and $ruo(\{C\},\ T_1)$ are calculated as 0.0769 and 0.4615. Since the next item is $\{B\}$, *N.Item* becomes $\{B\}$. Furthermore, *N.Idx* becomes 1 because the corresponding entry of GUO-IL($\{B\}$) is at the first position. Therefore, the entry, $<B$, 0.0769, 0.4615, 1 $>$, is added to a set of entries of GUO-IL($\{C\}$). The remaining item $\{A\}$ is handled in the same manner. The process for T_1 is illustrated in Fig. 5 (a). Then, let us consider processing T_3 after T_1 and T_2 are completely handled. Since GUO-IL($\{B\}$) already exists, information obtained in the same way as above is stored in a new quadruple entry, which is inserted into GUO-IL($\{B\}$) as the second entry. Hence, when the next item $\{G\}$ is read, *N.item* and *N.Idx* are set to $\{B\}$ and 2. This process is presented in Fig. 5 (b). The set of GUO-ILs configured after scanning 10 transactions in the revised database is shown in Fig. 6.

Transaction	A revised set of items (*name, utility*)	TU
T_1	$(A,6)\ (C,1)\ (B,6)$	13
T_2	$(D,12)\ (F,8)$	20
T_3	$(G,6)\ (B,8)$	14
T_4	$(A,9)(F,4)(C,4)$	21
T_5	$(D,6)\ (B,4)$	12
T_6	$(D,6)\ (C,1)\ (B,8)$	15
T_7	$(G,6)\ (F,4)$	10
T_8	$(A,9)\ (C,5)\ (B,2)$	16
T_9	$(G,12)\ (F,4)$	16
T_{10}	$(D,12)\ (C,5)\ (B,4)$	21

FIGURE 4. Example revised quantitative database.

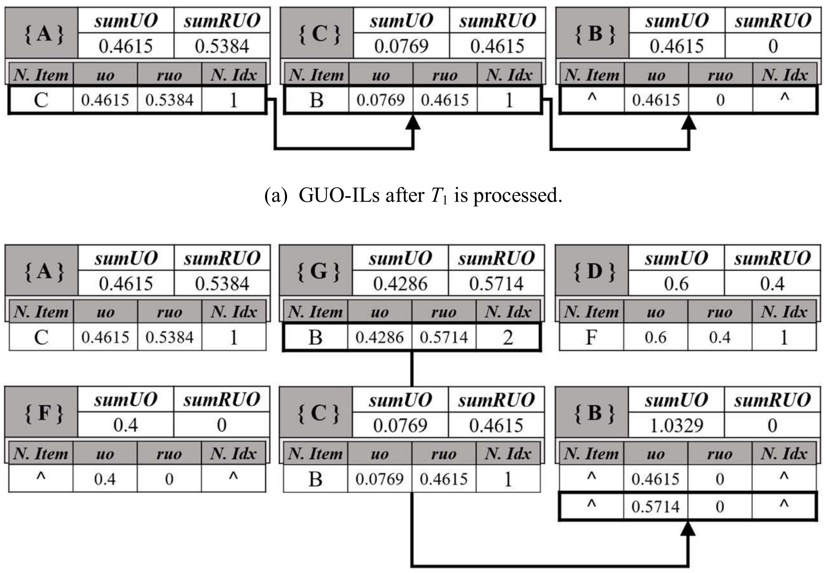
D. NOVEL METHOD FOR MINING HIGH UTILITY OCCUPANCY PATTERNs WITH INDEXED LIST STRUCTURE

This subsection introduces novel mining techniques utilizing the proposed GUO-IL and CUO-IL. The use of an upper bound affects the efficiency of an algorithm because it allows unnecessary pattern generation to be avoided. If a miner considers all combinable patterns in a database, it is exceedingly inefficient. For this reason, an upper bound, which allows the generation of unpromising patterns to be skipped, is a significant concern. We adopt several pruning strategies using the established upper bounds [17], [50]. The first strategy is to examine whether a pattern satisfies the constraint on support. According to the anti-monotone property, we have that the pattern whose support is not greater than a minimum support threshold cannot be extended to a high utility occupancy pattern. Then, a loose upper bound of the pattern is computed, and it is compared to a minimum utility occupancy threshold.

Definition 11: Suppose that there is a pattern X in *QDB*, the loose upper bound utility occupancy of X throughout *QDB* is denoted as $lubuo(X)$. The equation for $lubuo(X)$ is as follows:

$$(X)\ = \sum\{uo(X,T_k) + ruo(X,T_k)\} \\ /\lceil minSup(QDB,\gamma)\rceil, X \subseteq T_k \cap T_k \in QDB \quad (12)$$

In fraction (12), the numerator is obtained by adding the aggregated information of X. If the sum of *sumUO* and *sumRUO* divided by $\lceil minSup\ (QDB,\gamma)\rceil$ is less than a minimum utility occupancy threshold, any super patterns need not be confirmed. Even if the strategy using *lubuo* value is passed, it is finally checked if $ubuo(X)$ is less than the threshold.



{A}	sumUO	sumRUO
	0.4615	0.5384

N. Item	uo	ruo	N. Idx
C	0.4615	0.5384	1

{C}	sumUO	sumRUO
	0.0769	0.4615

N. Item	uo	ruo	N. Idx
B	0.0769	0.4615	1

{B}	sumUO	sumRUO
	0.4615	0

N. Item	uo	ruo	N. Idx
^	0.4615	0	^

(a) GUO-ILs after T_1 is processed.

{A}	sumUO	sumRUO
	0.4615	0.5384

N. Item	uo	ruo	N. Idx
C	0.4615	0.5384	1

{G}	sumUO	sumRUO
	0.4286	0.5714

N. Item	uo	ruo	N. Idx
B	0.4286	0.5714	2

{D}	sumUO	sumRUO
	0.6	0.4

N. Item	uo	ruo	N. Idx
F	0.6	0.4	1

{F}	sumUO	sumRUO
	0.4	0

N. Item	uo	ruo	N. Idx
^	0.4	0	^

{C}	sumUO	sumRUO
	0.0769	0.4615

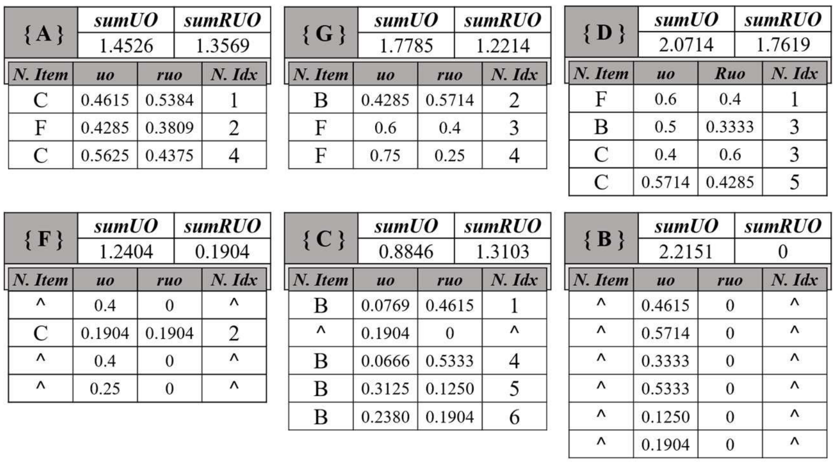
N. Item	uo	ruo	N. Idx
B	0.0769	0.4615	1

{B}	sumUO	sumRUO
	1.0329	0

N. Item	uo	ruo	N. Idx
^	0.4615	0	^
^	0.5714	0	^

(b) GUO-ILs after up to T_3 is processed.

FIGURE 5. **Generation of GUO-ILs from T_1 to T_3.**



{A}	sumUO	sumRUO
	1.4526	1.3569

N. Item	uo	ruo	N. Idx
C	0.4615	0.5384	1
F	0.4285	0.3809	2
C	0.5625	0.4375	4

{G}	sumUO	sumRUO
	1.7785	1.2214

N. Item	uo	ruo	N. Idx
B	0.4285	0.5714	2
F	0.6	0.4	3
F	0.75	0.25	4

{D}	sumUO	sumRUO
	2.0714	1.7619

N. Item	uo	Ruo	N. Idx
F	0.6	0.4	1
B	0.5	0.3333	3
C	0.4	0.6	3
C	0.5714	0.4285	5

{F}	sumUO	sumRUO
	1.2404	0.1904

N. Item	uo	ruo	N. Idx
^	0.4	0	^
C	0.1904	0.1904	2
^	0.4	0	^
^	0.25	0	^

{C}	sumUO	sumRUO
	0.8846	1.3103

N. Item	uo	ruo	N. Idx
B	0.0769	0.4615	1
^	0.1904	0	^
B	0.0666	0.5333	4
B	0.3125	0.1250	5
B	0.2380	0.1904	6

{B}	sumUO	sumRUO
	2.2151	0

N. Item	uo	ruo	N. Idx
^	0.4615	0	^
^	0.5714	0	^
^	0.3333	0	^
^	0.5333	0	^
^	0.1250	0	^
^	0.1904	0	^

FIGURE 6. **A complete set of GUO-ILs.**

The pruning techniques are applied in the above order, and HUOMIL effectively avoids exhausting generation tasks by the strategies. Note that the *lubuo* value is the upper bound first designed in [50], but the *lubuo* in the study considers items in transactions to which support-based pruning is not applied, rather than revised transactions.

Lemma 1: For two patterns that satisfy $X \subset Y$, if the $lubuo(X)$ is smaller than the minimum utility occupancy threshold, any Y can be pruned without confirmation of $ubuo(X)$.

Rationale: Let γ be a minimum support threshold, and δ be a minimum utility occupancy threshold. $lubuo(X)$, the loose upper bound of X, is expressed as follows: $\{sumUO(X) + sumRUO(X)\}/\lceil\gamma\rceil$. This formula is expanded to $[\sum_{T_k \in T^X}\{uo(X, T_k) + ruo(X, T_k)\}]/\lceil\gamma\rceil$. For any super pattern Y of X, $uo(Y) = \sum_{T_l \in T^Y} uo(Y, T_l)/sup(Y)$. Since $uo(Y, T_l) \le uo(X, T_l) + ruo(X, T_l)$ in each T_l ($T_l \in T^Y$), $[\sum_{T_l \in T^Y}\{uo(X, T_l) + ruo(X, T_l)\}]/sup(Y)$ is greater than or equal to $uo(Y)$. When $\lceil\gamma\rceil \le sup(Y)$, we have:

$$\frac{\sum\limits_{T_l \in T^Y}\{uo(X, T_l) + ruo(X, T_l)\}}{sup(Y)} \le \frac{\sum\limits_{T_l \in T^Y}\{uo(X, T_l) + ruo(X, T_l)\}}{\lceil\gamma\rceil}. \tag{13}$$

$$\Longrightarrow \frac{\sum_{T_l \in T^Y}\{uo(X, T_l) + ruo(X, T_l)\}}{\lceil\gamma\rceil} \le \frac{\sum_{T_k \in T^X}\{uo(X, T_k) + ruo(X, T_k)\}}{\lceil\gamma\rceil} \quad (T^Y \subseteq T^X). \tag{14}$$

$$\Longrightarrow uo\left(Y\right) \leq \frac{\sum_{T_k \in T^X} \left\{uo\left(X, T_k\right) + ruo\left(X, T_k\right)\right\}}{\lceil \gamma \rceil}. \quad (15)$$

Therefore, *lubuo*(X) is always greater than or equal to the utility occupancy of Y. In addition, for $\forall T_k (\in T^X)$, *ubuo*(X) is calculated when the numerator of the middle term in (15) is substituted to $\sum_{T_k \in T^X}^{Top\lceil\gamma\rceil} \{uo\left(X, T_k\right) + ruo\left(X, T_k\right)\}$, where Top $\lceil\gamma\rceil$ refers to the top $\lceil\gamma\rceil$ in the sum of $uo\left(X, T_k\right) + ruo\left(X, T_k\right)$. Because the numerator of *ubuo*(X) is not greater than that of *lubuo*(X), they are expressed by the following inequality.

$$\frac{\sum_{T_k \in T^X}^{Top\lceil\gamma\rceil} \left\{uo\left(X, T_k\right) + ruo\left(X, T_k\right)\right\}}{\lceil \gamma \rceil} \leq \frac{\sum_{T_k \in T^X} \left\{uo\left(X, T_k\right) + ruo\left(X, T_k\right)\right\}}{\lceil \gamma \rceil}. \quad (16)$$

According to the (16), if *lubuo*(X) is less than δ, then *ubuo*(X) is also always less than δ. Therefore, no pattern loss occurs even if any super pattern Y pruned by the loose upper bound is not inspected by *ubuo*(X).

In the previous studies, the utility list-based vertical data structure that manages *tid* for each transaction tries to find identical transactions. This vertical search is quite inefficient when there are many transactions where items appear and/or the length of patterns is potentially long. On the other hand, HUOMIL based on an indexed list can directly access the next item and its entry index without unnecessary comparisons. Therefore, it is possible to mine long patterns efficiently because it significantly reduces the comparison overhead.

Lemma 2: During the mining process, the number of operations required by an indexed list-based algorithm, $Alg^{IdxList}$, is always smaller than a list-based algorithm, Alg^{List}.

Rationale: For a quantitative database with n distinct items, let IL be a set of n global indexed lists for a 1-itemset, and L be a set of n global utility lists for a 1-itemset. The two sets are represented by $\{IL_1, IL_2, \ldots, IL_n\}$ and $\{L_1, L_2, \ldots, L_n\}$. The kth ($1 \leq k \leq n$) elements of IL and L have the same number of entries because they are constructed by scanning the identical quantitative database. Thus, it is expressed as $|IL_k| = |L_k|$. In addition, when IL_k and L_k is selected as a prefix for pattern extension, they create up to $(n - k)$ conditional data structures. We verify two possible cases by comparing the operations of $Alg^{IdxList}$ and Alg^{List}. The first case is when the kth elements of IL and L are prefix, and the number of elements that can be combined with it is $(n-k)$. Alg^{List} requires $\{2(|L_k|+|L_l|)-1\}\times(n-k)$ operations due to a two-way comparison method. On the other hand, $Alg^{IdxList}$ takes $|L_k|\times(n - k)$ because it only accesses the entry of connected IL_l. The second case is that the number of L_l bondable to L_k is less than $(n - k)$. No matter how many L_l are, Alg^{List} must navigate to L_n. Hence, it needs $\{2(|L_k| + |L_l|)-1\}\times(n - k)$. $Alg^{IdxList}$, however, approaches only combinable L_l with its index information. Therefore, $|L_k|\times p$ ($p< (n - k)$) is taken. This is true even if a pattern with a length of 2 or more is selected as a prefix in addition to 1-itemset.

{GUO-IL(i_1), …, GUO-IL(i_p), …, GUO-IL(i_n)}, which is a set of GUO-ILs configured after database scans, becomes a material performing the first mining. HUOMIL chooses a prefix GUO-IL(i_p) ($1 \leq p \leq n$), then visits the entries, $\{e^1(i_p), e^2(i_p), \ldots, e^m(i_p)\}$. In this step, if $\{i_p\}$ has less support than *minSup*(QDB, γ), the prefix is moved to $\{i_{p+1}\}$. Afterward, the prefix is determined by comparing its *uo* and *minUtilOcc*. The remaining two upper bounds are calculated and checked before $\{i_p\}$ is extended to longer ones. Starting with $e^1(i_p)$, all entries associated with $e^1(i_p)$ are tracked, which are assumed to be $\{e^1(i_p), e(i_{p+1}), \ldots, e(i_{n-1}), e(i_n)\}$. The items, connected entries, are combined with $\{i_p\}$, thereby creating CUO-IL($i_p \cup i_{p+1}$), …,CUO-IL($i_p \cup i_{n-1}$), and CUO-IL($i_p \cup i_n$). Moreover, each new quadruple entry is inserted into the corresponding CUO-IL($i_p \cup i_k$) ($p < k \leq n$) as follows: $<$ $i_p \cup i_{k+1}$, $e^1(i_p).uo+$ $e(i_k).uo$, $e(i_k).ruo$, $|$ CUO-IL($i_p \cup i_{k+1}$)$|$ $>$. In the case of the aggregated information, $e^1(i_p).uo+e(i_k).uo$ and $e(i_k).ruo$ are added to the previous aggregated information of CUO-IL($i_p \cup i_k$). Note that $\{i_p \cup i_{p+1}\}$ is chosen as a prefix for the next mining process using the above CUO-ILs.

For example, Fig. 7 shows an example mining process to generate 2-itemsets from a set of GUO-ILs. It is assumed that $\{A\}$ is selected as a prefix, and the two thresholds are 2 and 0.65, respectively. Since *sup*($\{A\}$) is 3, the mining continues. The *uo*($\{A\}$)=1.4526/30.4842, so it is a high utility occupancy pattern. In addition, both (1.4526+1.3569)/3 and *ubuo*($\{A\}$) are greater than *minUtilOcc*. First, $e^1(\{A\})$ is chosen, which is $<C$, 0.4615, 0.5384, 1 $>$. Then, *base-list* for prefix $\{A\}$, denoted as CUO-IL($\{A\_\}$), is generated, and 0.4615 is stored in the *base-list*. After that, starting with $e^1(\{A\})$, another connected entry $e^1(\{C\})$ is found. Thus, the created CUO-IL($\{\_C\}$) includes $<$ B, 0.5384, 0.4615, 1$>$. The *sumUO* and *sumRUO* of CUO-IL($\{_C\}$) accumulate 0.5384 and 0.4615, respectively. Note that the first entry of the *base-list* points $e^1(\{\_C\})$. Next, $\{B\}$ is combined with $\{A\}$ because $\{B\}$ is connected to $e^1(\{C\})$. This process is performed in the same manner. If this task is completed, $e^2(\{A\})$ and $e^3(\{A\})$ are also processed next.

However, when k-itemset ($k \geq 3$) is generated in this way, we obtain an inaccurate utility occupancy. The reason is that the prefix utility occupancy was duplicated when combining k-itemsets with the same prefix. To accurately compute utility occupancy, an indexed list, called a *base-list*, is created for each mining step. It then stores prefix utility occupancy and is connected to the corresponding entry. The connected entries correspond to a particular transaction, which means that the utility occupancy of the prefix is different for each transaction. Therefore, HUOMIL calculates *uo* of $\{P_1 \cup P_2\}$ by subtracting the prefix utility occupancy from the sum of the *uo* values of P_1 and P_2, where P_1 and P_2 are prefix pattern and a pattern to be combined with P_1, respectively. In this way, CUO-IL operates on a method that interacts with a *base-list* for accurate calculation. Thus, it is clearly distinguished from GUO-IL.

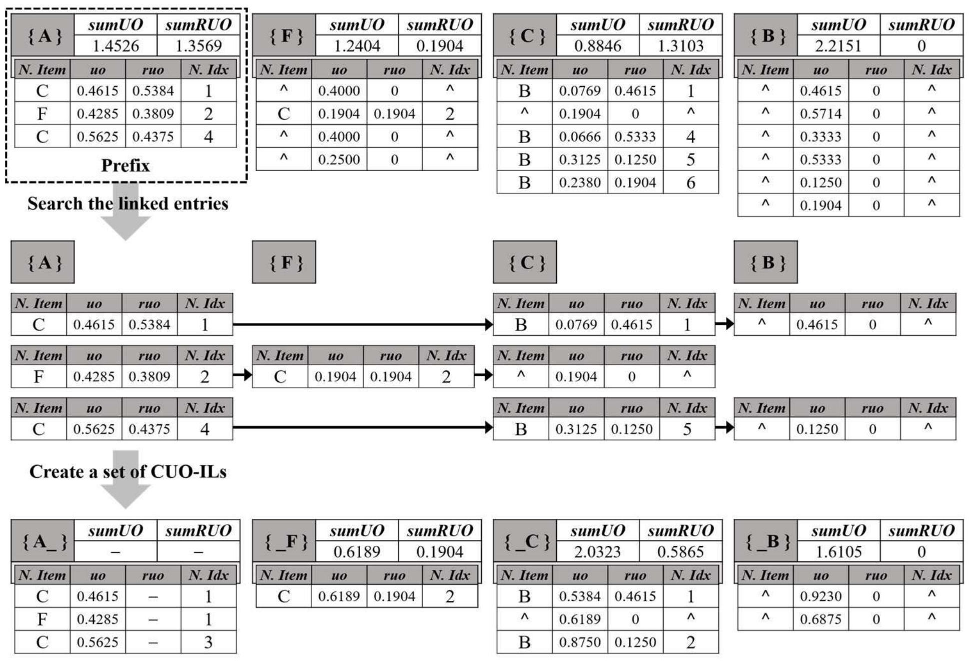


FIGURE 7. Construction of CUO-ILs when a prefix is {A}.

For example, Consider the generation 3-itemsets with prefix $\{A, F\}$. For an easy explanation, $\{A, F\}$ is expressed as $\{AF\}$. $sup(\{AC\})$ is 1, which is less than 3, so it is pruned. Fig. 8 illustrates that $\{ACB\}$ is generated by combining $\{AC\}$ and $\{AB\}$. Thus, CUO-IL($\{\_C\}$), whose name is $\{AC\}$, is a prefix. It can be seen that $e^1(\{\_C\})$ points $e^1(\{\_B\})$, CUO-IL($\_C$) with prefix $\{AC\}$ is generated and inserted into a set of CUO-ILs for the next mining level. A new quadruple entry to be added to the new CUO-IL($\_B$) includes the information as follows: <^, 0.5384+0.9230−0.4615, 0, ^>. The prefix utility occupancy, 0.4615, is referred from the *base-list* of CUO-IL($A\_$) at the previous mining level. In the same manner, CUO-IL($\_B$) with prefix $\{AC\_\}$ has thus two quadruple entries. Moreover, CUO-IL($AC\_$) is presented at the bottom of Fig. 8.

E. DESCRIPTION OF THE PROPOSED ALGORITHM

In this part, we detail the proposed HUOMIL algorithm, which employs the two efficient indexed list structures (GUO-IL and CUO-IL). HUOMIL performs three major procedures to discover high utility occupancy patterns. Algorithm 1 describes the main algorithm that finally returns the resulting patterns to a user, including the first database scan. Transactions in a quantitative database, *QDB*, are sequentially processed, and the items in each transaction are also scanned (Lines 01–06). In this process, support for all items existing in *QDB* is investigated (Lines 03–04). Two predefined threshold ratios are converted into minimum support and utility occupancy thresholds, which are denoted as $minSup(QDB, \gamma)$ and $minUtilOcc(\delta)$ (Lines 07 and 08). Among all items, I^{all}, items whose $sup(i)$ ($i \in I^{all}$) is greater than or equal to $minSup(QDB, \gamma)$ are included in I (Line 09). After that, the total order is decided by sorting I in support-ascending order (Line 10). The database is scanned again by Algorithm 2, *Construct*, and a set of GUO-ILs of items in I is acquired (Line 11). The set is passed to Algorithm 3, *Mine*, which returns a set of resulting patterns (Line 12). Finally, we return the high utility occupancy patterns (Line 13).

Algorithm 1 HUOMIL

```
Input: A quantitative database, $QDB$; an external utility
       table, $EU\text{-}table$; a support threshold, $\gamma$; a utility
       occupancy threshold, $\delta$
Output: A set of resulting patterns, $ResultSet$
Variable: A set of items in the database, $I^{all}$; a set of items
          satisfying support constraint, $I$; a set of GUO-
          ILs, $GUO$
01. for each transaction $T_k \in QDB$ do
02.     for each item $i \in T_k$ do
03.         $I^{all} \leftarrow i$;
04.         Increase $Sup(i)$ by 1 ;
05.     end for
06. end for
07. $minSup \leftarrow |QDB| \times \gamma$ ;
08. $minUtilOcc \leftarrow \delta$ ;
09. $I \leftarrow \{i \mid i \in I^{all} \wedge Sup(i) \geq minSup\}$ ;
10. Sort $I$ in support-ascending order ;
11. $GUO \leftarrow Construct\ (QDB, EU\text{-}table, I)$ ;
12. $ResultSet \leftarrow Mine\ (\emptyset, GUO, minUtilOcc)$;
13. Return $ResultSet$;
```

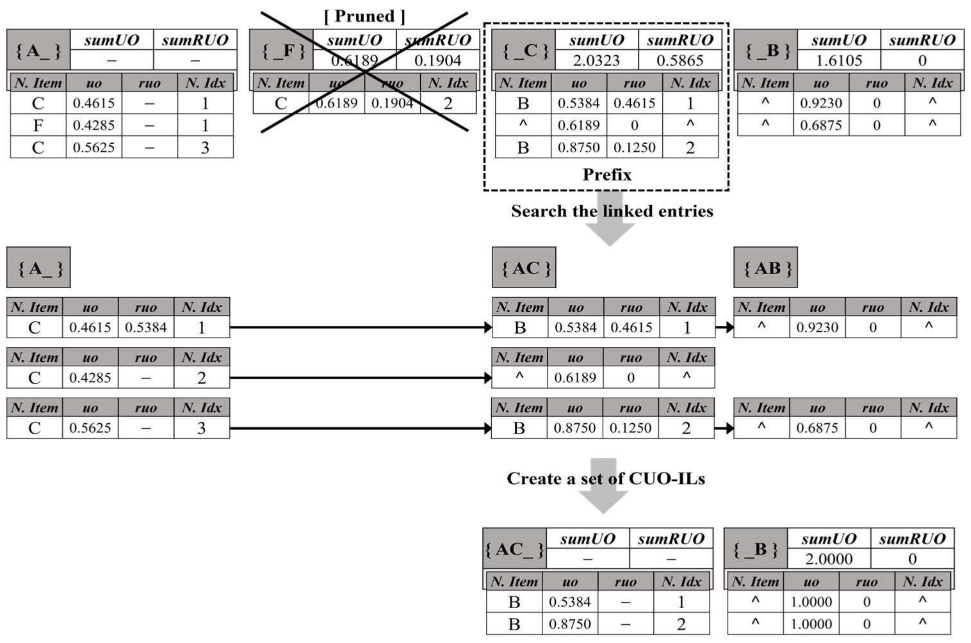


FIGURE 8. Construction of CUO-ILs when a prefix is {AF}.

The details of organizing a complete set of GUO-ILs with a database scan are shown in Algorithm 2. A set for managing GUO-IL for each item is initialized (Line 01). All transactions in the database are sequentially traversed once again (Lines 02–18). Each transaction is sorted in reverse of the total order (Line 03). At this time, variables related to index information and *ruo* value are defined. Especially the initial values of the index information, *N.Item* and *N.Idx*, are NULL (Lines 04 and 05). Only items included in I are visited in reverse order (Lines 06–17). If there is no GUO-IL of i, a new one is created and inserted into the global set (Lines 07–09). Otherwise, a new quadruple entry is added to GUO-IL of i (Line 10). The utility occupancy of i, *uo*, is obtained by using item and transaction utilities (Line 11). Eventually, the entry includes quadruple one: <*N.Item*, *uo*, *ruo*, *N.Idx*> (Line 12). After that, the *uo* value is accumulated in the *ruo* (Line 13). In addition, *N.Item* and *N.Idx* are changed to the identifier of i and the number of entries in GUO-IL(i) for the next item to be processed (Lines 14 and 15). At last, the aggregated information, *sumUO* and *sumRUO*, is updated by adding *uo* and *ruo* (Line 16). The complete set of GUO-ILs is transferred to the main algorithm (Line 19).

Algorithm 3 illustrates the mining method of HUOMIL, which adopts a prefix selection and extension manner to generate k-itemsets ($k \geq 2$). The set CUO_k generated in the mining step for k-itemsets is used to make CUO_{k+1}, a set of CUO-ILs of $(k+1)$-itemsets (Line 01). Each CUO-IL(P), the CUO-IL of a pattern P, in CUO_k is selected as a prefix (Lines 02–31). If the support of the prefix is greater than or equal to $minSup(QDB, \gamma)$, the following processes are conducted (Line 03–27). It is identified whether P is a high utility occupancy pattern (Line 04–06). Separately, to apply the rest of the pruning strategies, it is confirmed that both $(sumUO+sumRUO)/\lceil minSup(QDB, \gamma)\rceil \geq minUtilOcc(\delta)$ and $ubuo(P) \geq minUtilOcc$ are satisfied (Lines 07–26 and 09–25). Note that the calculation of $ubuo(P)$ follows definition 9. Each entry of CUO-IL(P), e, is visited in turn (Lines 11–24). In the process of CUO_{k+1} generation, a base-list is first generated, and it stores the utility of P as a prefix utility (Line 12). Then, the other entries connected to e, ce, are tracked until it no longer exists, and combined with e (Lines 13–23). A new CUO-IL of a pattern extended from P is generated if it does not exist in CUO_{k+1} (Lines 14–16), and a new quadruple entry, ne, is inserted into there (Line 17). To accurately calculate utility occupancy, approach the *uo* of the prefix from the base-list of CUO_k (Line 18). The sum of *uo* values of e and ce is reduced by the *uo* of the prefix, and it becomes *uo* of ne (Line 19). The *ruo* of ne is *ruo* of ce(Line 20). The aggregated information of the CUO-IL is then updated (Line 21). At last, the previously generated entry and the currently processed entry are connected based on index information (Lines 21 and 22). The mining process is repeatedly called in the DFS-based manner if the size of CUO_{k+1} is not zero (Line 28). In this process, the set of results is continuously accumulated (Line 29).

Let the database size and the number of unique items in the database be m and n, respectively. It costs $O(mn)$ to calculate the support of all items and $O(nlog_2n)$ to sort items in support ascending order. Thus, 1st database scan consumes $O(mn+nlog_2n)$ to compute its own total order. It costs $O(nlog_2n)$ to sort items in each transaction and $O(n)$ to insert all items into the GUO-ILs. Since m transactions are

```
Algorithm 2 Construct
Input: A quantitative database, $QDB$; an external utility
         table, $EU\text{-}table$; a set of items according to
         total order, $I$
Output A set of GUO-ILs, $GUO$
01.  Initialize $GUO$ ;
02.  for each transaction $T_k \in QDB$ do
03.    Revise $T_k$ to $T'_k$ ;
04.    $ruo \leftarrow 0$ ;
05.    $N.Item, N.Idx \leftarrow 0$ ;
06.    for each item $i \in I$ and $i \in T'_k$ do
07.      if $GUO(i)$ does not exist in $GUO$then
08.        Create $GUO(i)$ and insert into $GUO$ ;
09.      end if
10.      Create a new quadruple entry $e$ in $GUO(i)$ ;
11.      $uo \leftarrow u(i, T_k) / tu(T_k)$ ;
12.      $e \leftarrow <N.Item, uo, ruo, N.Idx>$ ;
13.      $ruo \leftarrow ruo + uo$;
14.      $N.Item \leftarrow i$;
15.      $N.Idx \leftarrow |GUO(i)|$ ;
16.      Update $sumUO$ and $sumRUO$ of $GUO(i)$ ;
17.    end for
18.  end for
19.  Return $GUO$;
```

processed, 2nd database scan spends O($mnlog_2n$) to construct GUO-ILs for all items. It costs O(m) to construct a CUO-IL for an itemset, and there can be at most 2^n-1 possible itemsets. Thus, the pattern extension process consumes O($m2^n$) to create CUO-ILs for all itemsets and extract all high utility occupancy patterns.

IV. PERFORMANCE EVALUATION

A. EXPERIMENTAL CONDITIONS AND DATASETS

We conducted extensive experiments to evaluate the efficiency of our algorithm. The comparison algorithms are HUOPM [17] and OCEAN [15]. These are state-of-the-art approaches that discover high utility occupancy patterns from a quantitative database. This means that the proposed HUOMIL algorithm and two comparison algorithms belong to the high utility occupancy pattern mining equally. OCEAN uses not only ascending order according to TWU, but also several pruning techniques. Meanwhile, HUOPM applies more efficient pruning strategies that are not used in OCEAN as well as adapts support-ascending order as a total order. HUOMIL leverages indexed list-based data structure, so that it efficiently performs the pattern extension during the mining process. In addition, it uses a strategy with aggregated information. We measured the runtime and peak memory usage of the above algorithms to evaluate the efficiency of HUOMIL by changing the pre-defined thresholds. All the above algorithms were implemented in C/C++ language. The runtime and memory tests were conducted on a PC with the following specifications: Intel Core i7-6700K CPU @ 4.00GHz, 32GB

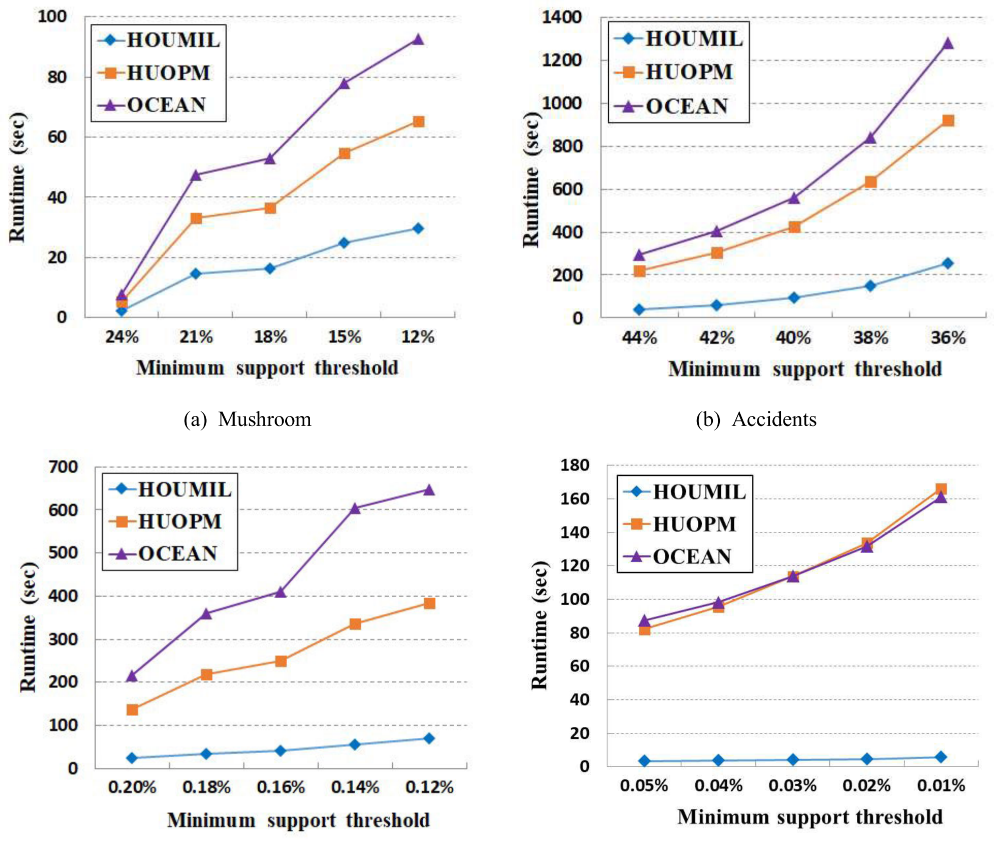
```
Algorithm 3 Mine
Input: A prefix in the previous mining level, $prefix$; a set of
         CUO-ILs for $k$-itemsets, $CUO_k$; a minimum utility
         occupancy threshold, $minUtilOcc$; a minimum
         support threshold, $minSup$
Output: A set of resulting patterns, $ResultSet$
Variable: A set of CUO-ILs for ($k$+1)-itemsets, $CUO_{k+1}$; a
         CUO-IL of a pattern $P$, $CUO(P)$; a loose upper
         bound of a pattern $P$, $lubuo(P)$; an upper bound of
         a pattern $P$, $ubuo(P)$
01.  Initialize $CUO_{k+1}$ ;
02.  for each CUO-IL $CUO(P) \in CUO_k$ do
03.    if $sup(P) \geq minSup$ then
04.      if ($CUO(P).sumUO/sup(P) \geq minUtilOcc$)then
05.        $ResultSet \leftarrow \{ResultSet\} \cup \{Prefix \cup P\}$ ;
06.      end if
07.      if ($lubuo(P) \geq minUtilOcc$)then
08.        $ubuo(P) \leftarrow UpperBound(CUO(P), minSup)$;
09.        if ($ubuo(P) \geq minUtilOcc$) then
10.          $Prefix \leftarrow \{Prefix \cup P\}$;
11.          for each entry $e \in CUO(P)$ do
12.            Create $Base\text{-}list$ for $Prefix$ in $CUO_{k+1}$ and
               Insert a new entry ;
13.            for each entry $ce$ connected with $e$ do
14.              if CUO-IL does not exist in $CUO_{k+1}$ then
15.                Create a new CUO-IL, $newCUO$, in $CUO_{k+1}$ ;
16.              end if
17.              Create a new entry $ne$ in $newCUO$;
18.              Find the entry $pe$ of $Base\text{-}list$ in $CUO_k$ ;
19.              $ne.uo \leftarrow e.uo + ce.uo - pe.uo$ ;
20.              $ne.ruo \leftarrow e.ruo$ ;
21.              Update $sumUO$ and $sumRUO$ of $newCUO$;
22.              Connect $ne$ to the previous entry ;
23             end for
24           end for
25         end if
26.      end if
27.    end if
28.    if ( $|CUO_{k+1}| \neq 0$ ) then
29.      $ResultSet \leftarrow \{ResultSet\} \cup \{Mine(Prefix, CUO_{k+1},$
         $minUtilOcc, minSup)\}$ ;
30.    end if
31.  end for
32.  Return $ResultSet$;
```

TABLE 1. Features of Real Datasets.

Type	Dataset (QDB)	$\lvert QDB \rvert$	$\lvert I \rvert$	$\lvert T \rvert^{AVG}$
(Dense) Real	Mushroom	8,124	119	23
	Accidents	340,183	942	33.808
(Sparse) Real	Retail	88,162	16,469	10.3
	Kosarak	36,869	41,270	7.2

memory, and Windows 10 operating system, and we performed these experiments in Visual Studio.

To conduct rigorous performance evaluation, we used real datasets, which have different characteristics. Table 1 and Table 2 indicate real datasets and synthetic datasets that are used in the experiments. The real datasets are benchmark



(a) Mushroom (b) Accidents

(c) Kosarak (d) Retail

FIGURE 9. Runtime results on real datasets at varying minimum support thresholds (δ =0.3).

TABLE 2. Features of Synthetic Datasets.

Type	Dataset (QDB)	x	|QDB|	|I|	$\|T\|^{AVG}$
(Sparse) Synthetic	T10I4DxK	200	200,000	870 (Set)	10 (Set)
		400	400,000		
		600	600,000		
		800	800,000		
		1,000	1,000,000		

datasets that are frequently used in pattern mining and are classified into two types, Dense and Sparse. The dense dataset contains a small number of definite items, while the average length of transactions is relatively long. This implies that items tend to appear in the same transaction. In contrast, the sparse dataset has a lot of distinct items and long transactions. Thus, there is a high probability that items appear in different transactions. Table 1 states the key characteristics of each real dataset, which includes the number of transactions (| QDB|), the number of items (|I|), and the average length of transactions ($|T|^{AVG}$). Meanwhile, in order for real datasets to fit the experiments, the internal and external utilities of items were randomly generated for each transaction, which was referenced from former utility-driven pattern mining studies [16], [17], [22], [44]. The ranges for internal and external utilities are 1 to 10 and 0.01 to 10.00, respectively. These datasets are available in FIMI repository (http://fimi.ua.ac.be/data/). Table 2 describes the features of the synthetic datasets that we used in the scalability test. They are grouped into T10I4DxK, where xK stands for the number of transactions. In other words, T10I4D200K has 200,000 transactions. The reason we used T10I4DxK is that it is suitable for assessing the scalability of algorithms because all features are the same except for the number of transactions.

B. TEST FOR VARIOUS MINIMUM SUPPORT THRESHOLDS

In order to confirm the efficiency of the algorithms as the support threshold, γ varies, and the utility occupancy threshold, δ, of all datasets was fixed at 0.3. Therefore, in this section, we assess their efficiency by comparing their execution time and memory usage of them by gradually decreasing γ.

In Figs. 9 (a) to (d), the runtime results are presented at varying support thresholds. As shown in Fig.9 (a), the execution time of the three algorithms is found to be similar to less than 10 seconds when γ is 24%. This is because a small number of patterns are processed from Mushroom dataset to the corresponding threshold. However, considering the runtime when γ is lowered to 12%, we can see that HUOMIL needs less time than others to handle many patterns. In detail,

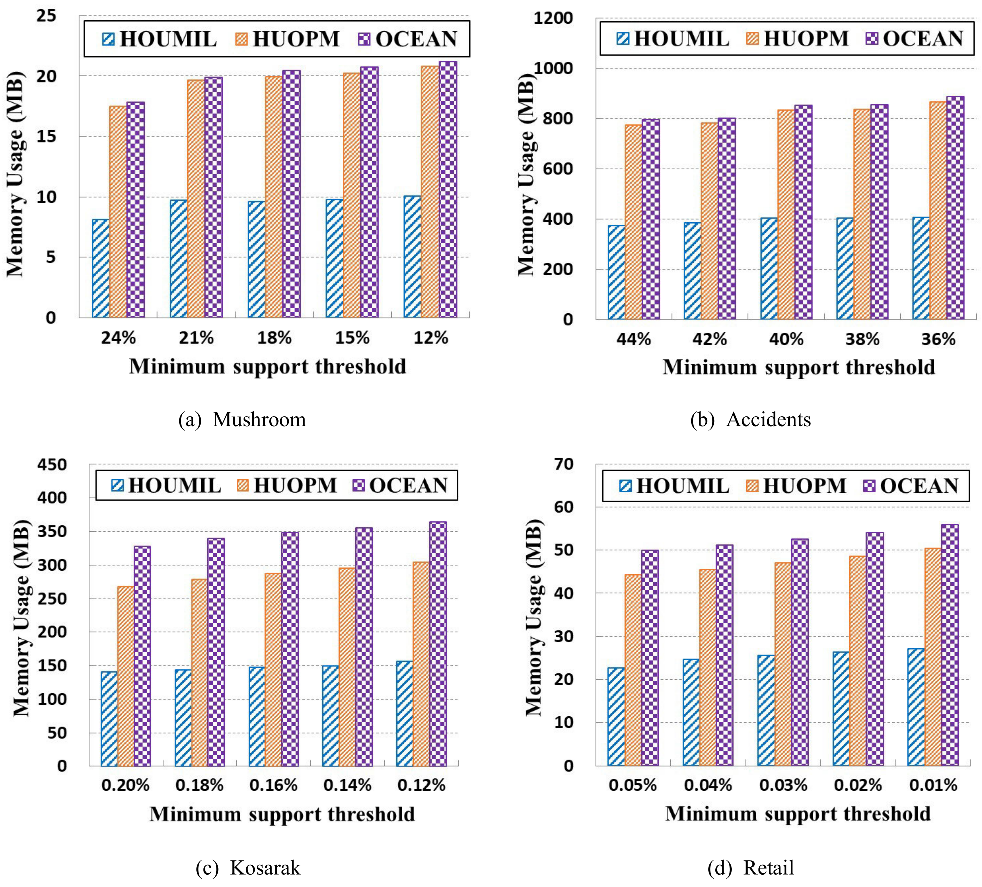


FIGURE 10. Memory usage results on real datasets at varying minimum support thresholds (δ =0.3).

it is approximately 3 and 2 orders of magnitude faster than OCEAN and HUOPM. In Accidents dataset, most algorithms spend quite a lot of time, and it is observed in Fig. 9 (b). OCEAN and HUOPM are greatly affected by γ in this situation. HUOMIL, on the other hand, is relatively stable to the reduction of γ, as opposed to others in the predefined ranges. In Fig. 9 (c) for Kosarak dataset, it can be seen that the runtime of OCEAN increases significantly after the 0.16% point, but at the same point, HUOPM has a rise smaller than that of OCEAN. However, HUOMIL increases slowly, and has a lower runtime than others. At last, Fig. 9 (d) indicates the results for Retail dataset. While the measurements of OCEAN and HUOPM increase steadily, HUOMIL performs mining with the least time. Although not identified in the graph, about 3.5 seconds were consumed at 0.05%, and about 6 seconds were measured at 0.01%. As a result, it is revealed that HUOMIL is more efficient than the comparative algorithms under the condition where the support threshold changes.

Figs. 10 (a) to (d) are the result of memory usage for each dataset captured in the tests performed in the previous runtime tests. In general, the lower the support threshold, the less the pruning effect can result in more pattern extension. This is why memory usage increases. In Fig. 10 (a), which shows the memory results for Mushroom dataset, the difference between OCEAN and HUOPM is slight, but they generally require about 50% more memory space than HUOMIL. This tendency similarly appears in Fig. 10 (b) for Accidents dataset. In Accidents, the memory usage of HUOMIL is half that of OCEAN and HUOPM. Fig. 10 (c) presents the results for Kosarak dataset. According to the memory usage on the graph, HUOPM uses less memory than OCEAN unlike the results on dense datasets due to the differences in the search space. Nevertheless, it is observed that HUOMIL has better memory efficiency. The size of Retail is smaller than that of Kosarak, it seems that memory usage is generally smaller than the figures shown in Fig. 10 (c). Based on Fig. 10 (d), HUOMIL always occupies less than 100MB in the given threshold range. HUOMIL includes quadruple entry with greater cardinality than the triple entry in OCEAN and HUOPM. However, we effectively addressed it by exploiting a sequence container, vector structure. Additionally, a specific structure for calculating *ubuo* value is situationally omitted because HUOMIL adopts a pruning technique based on the aggregated information.

C. TEST FOR VARIOUS MINIMUM UTILITY OCCUPANCY THRESHOLDS

In this subsection, we evaluate the efficiency of the algorithms under the condition where the occupancy threshold,

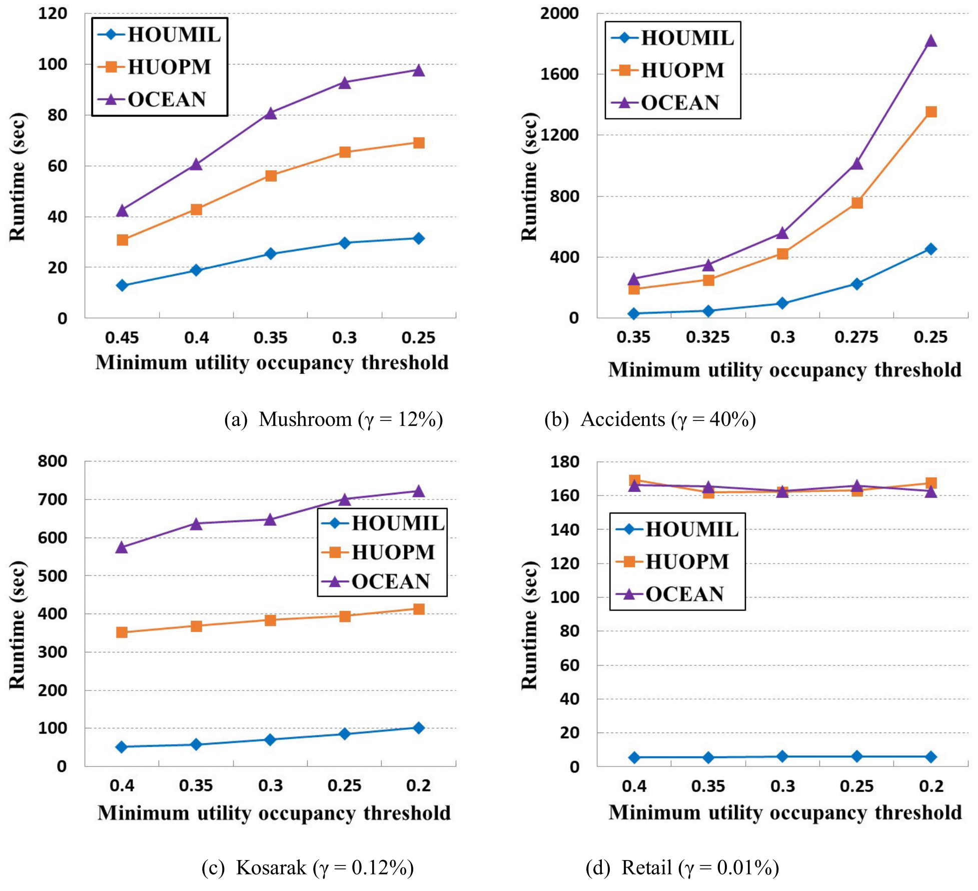
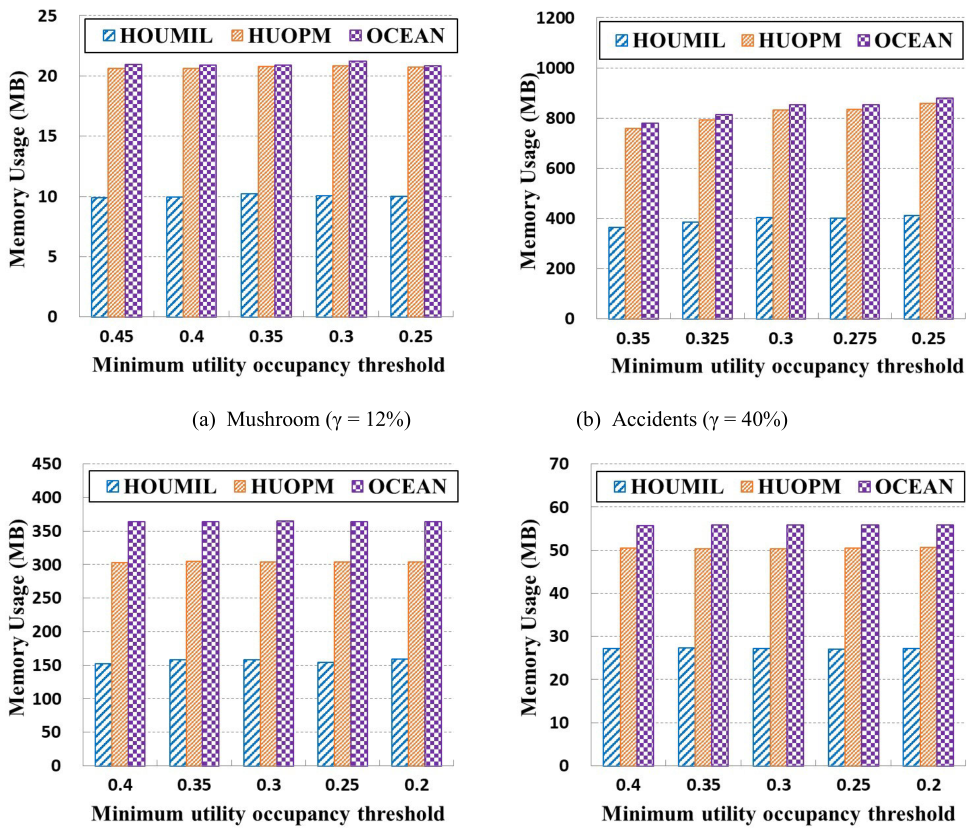


FIGURE 11. **Runtime results on real datasets at varying minimum utility occupancy threshold.**

δ, decreases. Note that the support threshold, γ, was fixed for each dataset, as shown in Figs. 12 (a)–(d).

As δ is lowered, the algorithms explore a wider search space. In experimental tests, the runtime of HUOMIL, measured from real datasets, is less than the comparative algorithms in all cases. This is caused by an efficient pattern extension method based on index information, even if patterns with long lengths occur. According to the results of Mushroom dataset in Fig. 11 (a), HUOPM is more quickly performed than OCEAN. The runtime of HUOMIL is also increasing, but it is quicker than the others. In Accidents dataset, the performance of the algorithms decreased rapidly because they react sensitively to varying δ. We can see this remarkable trend in Fig. 11 (b). Nonetheless, it is not only the fastest in all sections but also has the lowest growth rate. Fig. 11 (c) shows the runtime results for Kosarak dataset. In the graph, the runtime of HUOMIL is 4 and 7 times faster than HUOPM and OCEAN. As shown in Fig. 11. (d) for Retail dataset, the execution times of HUOPM and OCEAN are almost similar, and there are sometimes overlapping points. In addition, the runtime of the algorithms is mostly constant regardless of δ, and it indicates that they are not significantly affected by δ. However, HUOMIL is the fastest to mine the patterns, and the runtime is always less than 20 seconds in the range. Consequently, based on the above results, the runtime performance of HUOMIL can be seen that it is always quicker than the comparative algorithms in any interval.

The memory efficiency of each algorithm is evaluated in this paragraph through tests measuring peak memory usage. Fig. 12 (a), where Mushroom dataset is a target, shows that each algorithm uses a certain level of memory space for all δ. Despite the similarity between OCEAN and HUOPM, HUOMIL uses less than half of that of others. In Fig. 12 (b), we can visually confirm that memory usage tends to rise gradually. This suggests that there is a lot of pattern extensions occur in the experimental setting. At that time, the specific measurements were as follows: HUOMIL was 410.848MB, HUOPM was 857.77MB, and OCEAN was 880.422MB. That is, HUOPM and OCEAN represent comparable performance, but HUOMIL is relatively better than them. Figs. 12 (c) and (d) demonstrate the difference between HUOPM and OCEAN in Kosarak and Retail. The memory usage of HUOPM and OCEAN is over 300 MB and 50 MB, while the memory usage of HUOMIL is half of that of the comparison algorithms. According to the previous results, it can be seen that the difference between HUOPM and OCEAN is clear. This is the result of efficient pruning



(a) Mushroom (γ = 12%) (b) Accidents (γ = 40%)

(c) Kosarak (γ = 0.12%) (d) Retail (γ = 0.01%)

FIGURE 12. Memory usage results on real datasets at varying minimum utility occupancy thresholds.

strategies of HUOPM and the effect of the sorting order. HUOMIL is effectively treated with the vector-based data structure for efficient memory usage, even though it has entries whose cardinality is larger than HUOPM. Additionally, *lubuo* is applied before the application of *ubuo*, which is tighter but involves unnecessary storage space. As a result, the proposed HUOMIL uses memory space more efficiently than comparative algorithms in an experimental environment where δ changes.

D. SCALABILITY TEST

In this experiment, we evaluate the effect of the number of transactions on the efficiency of HUOMIL and compare ours with the comparative algorithms. To observe the scalability for increasing the number of transactions, the remaining characteristics of the synthetic datasets, except for the number of transactions, were fixed with the same number. Moreover, two thresholds, γ, and δ, were also set to 0.05% and 0.25. This test targeted five synthetic datasets whose database size increases from 200K to 1000K, with each dataset increasing in size by 200K. In general, as the number of transactions grows, the resource consumption of the algorithms becomes large. The reason is that runtime and memory usage rely on the amount of data to be processed. Although HUOMIL, HUOPM, and OCEAN tend to follow this trend, the results in Fig. 13 (a) show that the growth rate of HUOMIL is the gentlest. When the value of the x-axis is 1000K, HUOMIL is about 13 times faster than HUOPM, and about 16 times faster than OCEAN. In Fig. 13 (b), the memory usage of each algorithm always increases as the volume of the dataset grows by 200K, among which HUOMIL has a relatively small variation. According to the results of runtime and memory footprint tests, the proposed algorithm is more stable than the latest algorithms in terms of scalability.

V. DISCUSSION

In this section, we discuss the limitations of the proposed HUOMIL approach and future directions. The proposed approach, HUOMIL, processes a given database and mines high utility occupancy patterns with efficient indexed list-based data structures. By maintaining index information on associated itemsets, HUOMIL only explores entries of the indexed lists connected to each other. We observed that the efficiency of HUOMIL is better than the list-based approaches, HUOPM and OCEAN, in terms of runtime, memory usage, and scalability. This was demonstrated by experimental results on real and synthetic datasets regardless of the given thresholds and the size of the datasets.

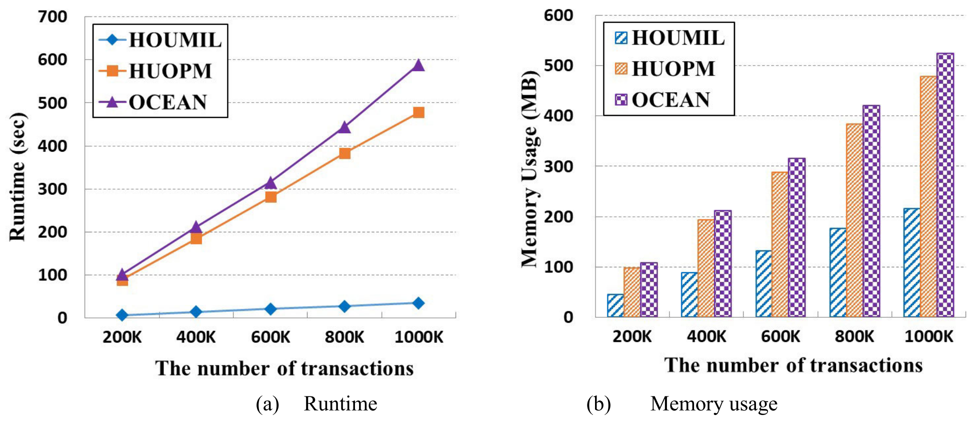


FIGURE 13. Runtime and Memory performance for synthetic datasets, T10I4DxK.

Despite the advantages of HUOMIL, there are several limitations. HUOMIL accesses the stacked data twice to consider only a set of optimized items. This can lead to a fatal problem for certain real-world applications, such as stream processing applications whose data incrementally occur. Furthermore, various types of data in the real world have some factors to be considered, but HUOMIL does not take into account them, such as noisy data [57], dynamic data [58], or temporal factors [33], [43]. On top of these limitations, the explosive increase in mining time interferes with the quick decision-making. For this reason, the mining performance needs to be improved, which is one of the main concerns in the pattern mining field.

The further directions based on the above limitations are as follows. Research, which covers effective responses to incremental data in dynamic databases, has been being studied. The approaches that have been proposed to process incremental data with a single scan or a small number of scans are still developing to improve their efficiency. Regarding the consideration of various types of data, a reasonable measure can be established in addition to utility and occupancy. For example, in study [56], a new measure provides useful insight by considering weight and occupancy. Finally, the mining efficiency can be improved by using an efficient data structure, changing sorting order, and applying pruning techniques to narrow wide search space. Apart from these, the combination with deep learning can also be one way, and the attempts have been studied [59], [60].

VI. CONCLUSION

In this paper, we proposed a novel approach called HUOMIL for efficiently extracting complete high utility occupancy patterns from a recorded quantitative database. The newly designed data structures, named GUO-IL and CUO-IL, retain index information of patterns in the form of a quadruple entry. According to the framework, each structure is connected to quickly perform the combination without a time-consuming search during the mining process. We addressed the limitations of the latest list-based approaches with efficient data structures and mining techniques. In addition, support and utility occupancy-driven upper bounds were used to reduce the search space of HUOMIL. We also adopted a loose upper bound pruning technique in order to avoid the computation required by the previous upper bound. To evaluate the efficiency of our method, we conducted performance tests with various real and synthetic datasets. The experimental results showed that our approach outperforms state-of-the-art methods in terms of execution time, memory usage, and scalability as two thresholds change

Meanwhile, data in the real world is accompanied by diverse factors depending on the collection environment. For example, there are temporal properties and data noise. This implies that various measurements can be combined with the occupancy concept to find valuable information. Therefore, in our future work, we plan to devise an approach to discover potentially significant patterns utilizing the existing occupancy-driven methods. In addition, we are scheduled to research the approach to dealing with incremental data in a dynamic database with novel indexed list-based data structures.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.
[2] W. Thurachon and W. Kreesuradej, "Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm," *IEEE Access*, vol. 9, pp. 55726–55741, 2021.
[3] Y. Xun, X. Cui, J. Zhang, and Q. Yin, "Incremental frequent itemsets mining based on frequent pattern tree and multi-scale," *Exp. Syst. Appl.*, vol. 163, Jan. 2021, Art. no. 113805.
[4] J. Adhikari, "Occupancy based pattern mining: Current status and future directions," *Int. J. Next Gener. Comput.*, vol. 11, no. 1, pp. 36–51, 2020.
[5] Z.-H. Deng, "Mining high occupancy itemsets," *Future Gener. Comput. Syst.*, vol. 102, pp. 222–229, Jan. 2020.
[6] L. Zhang, P. Luo, L. Tang, E. Chen, Q. Liu, M. Wang, and H. Xiong, "Occupancy-based frequent pattern mining," *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 2, pp. 1–33, 2015.
[7] H. Cheng, M. Han, N. Zhang, X. Li, and L. Wang, "A survey of incremental high-utility pattern mining based on storage structure," *J. Intell. Fuzzy Syst.*, vol. 41, no. 1, pp. 841–866, Aug. 2021.
[8] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1306–1327, Apr. 2021.

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

[10] J. C.-W. Lin, P. Fournier-Viger, V. S. Tseng, and P. S. Yu, "IEEE access special section editorial: Utility-pattern mining: Theoretical analytics and applications," *IEEE Access*, vol. 9, pp. 16604–16607, 2021.

[11] J. C. Lin, Y. Li, P. Fournier-Viger, Y. Djenouri, and J. Zhang, "Efficient chain structure for high-utility sequential pattern mining," *IEEE Access*, vol. 8, pp. 40714–40722, 2020.

[12] L. T. T. Nguyen, D.-B. Vu, T. D. D. Nguyen, and B. Vo, "Mining maximal high utility itemsets on dynamic profit databases," *Cybern. Syst.*, vol. 51, no. 2, pp. 140–160, Feb. 2020.

[13] J. M.-T. Wu, G. Srivastava, M. Wei, U. Yun, and J. C.-W. Lin, "Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework," *Inf. Sci.*, vol. 553, pp. 31–48, Apr. 2021.

[14] T. Mai, L. T. T. Nguyen, B. Vo, U. Yun, and T.-P. Hong, "Efficient algorithm for mining non-redundant high-utility association rules," *Sensors*, vol. 20, no. 4, p. 1078, Feb. 2020.

[15] B. Shen, Z. Wen, Y. Zhao, D. Zhou, and W. Zheng, "OCEAN: Fast discovery of high utility occupancy itemsets," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*, 2016, pp. 354–365.

[16] M. Liu and J.-F. Qu, "Mining high utility itemsets without candidate generation," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 55–64.

[17] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, "HUOPM: High-utility occupancy pattern mining," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1195–1208, Mar. 2020.

[18] L. T. T. Nguyen, P. Nguyen, T. D. D. Nguyen, B. Vo, P. Fournier-Viger, and V. S. Tseng, "Mining high-utility itemsets in dynamic profit databases," *Knowl.-Based Syst.*, vol. 175, pp. 130–144, Jul. 2019.

[19] J. M.-T. Wu, G. Srivastava, J. C.-W. Lin, Y. Djenouri, M. Wei, R. M. Parizi, and M. S. Khan, "Mining of high-utility patterns in big IoT-based databases," *Mobile Netw. Appl.*, vol. 26, no. 1, pp. 216–233, Feb. 2021.

[20] U. Yun, H. Kim, T. Ryu, Y. Baek, H. Nam, J. Lee, B. Vo, and W. Pedrycz, "Prelarge-based utility-oriented data analytics for transaction modifications in Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17333–17344, Dec. 2021.

[21] C. Liu and C. Guo, "Mining top-N high-utility operation patterns for taxi drivers," *Exp. Syst. Appl.*, vol. 170, May 2021, Art. no. 114546.

[22] C. Lee, Y. Baek, T. Ryu, H. Kim, H. Kim, J. Chun-Wei Lin, B. Vo, and U. Yun, "An efficient approach for mining maximized erasable utility patterns," *Inf. Sci.*, vol. 609, pp. 1288–1308, Sep. 2022.

[23] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*, 2005, pp. 689–695.

[24] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.

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

[26] P. Fournier-Viger, C. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Proc. Int. Symp. Methodologies Intell. Syst.*, 2014, pp. 83–92.

[27] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Exp. Syst. Appl.*, vol. 42, no. 5, pp. 2371–2381, 2015.

[28] S. Krishnamoorthy, "HMiner: Efficiently mining high utility itemsets," *Exp. Syst. Appl.*, vol. 90, pp. 168–183, Dec. 2017.

[29] H. Ryang and U. Yun, "Indexed list-based high utility pattern mining with utility upper-bound reduction and pattern combination techniques," *Knowl. Inf. Syst.*, vol. 51, no. 2, pp. 627–659, 2017.

[30] U. Yun, H. Nam, G. Lee, and E. Yoon, "Efficient approach for incremental high utility pattern mining with indexed list structure," *Future Gener. Comput. Syst.*, vol. 95, pp. 221–239, Jun. 2019.

[31] H. Nam, U. Yun, E. Yoon, and J. C. W. Lin, "Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions," *Inf. Sci.*, vol. 529, pp. 1–27, Aug. 2020.

[32] H. Kim, U. Yun, Y. Baek, J. Kim, B. Vo, E. Yoon, and H. Fujita, "Efficient list based mining of high average utility patterns with maximum average pruning strategies," *Inf. Sci.*, vol. 543, pp. 85–105, Jan. 2021.

[33] J. Kim, U. Yun, H. Kim, T. Ryu, J. C. Lin, P. Fournier-Vier, and W. Pedrycz, "Average utility driven data analytics on damped windows for intelligent systems with data streams," *Int. J. Intell. Syst.*, vol. 36, no. 10, pp. 5741–5769, Oct. 2021.

[34] J. M.-T. Wu, Q. Teng, J. C.-W. Lin, and C.-F. Cheng, "Incrementally updating the discovered high average-utility patterns with the pre-large concept," *IEEE Access*, vol. 8, pp. 66788–66798, 2020.

[35] J. C.-W. Lin, M. Pirouz, Y. Djenouri, C.-F. Cheng, and U. Ahmed, "Incrementally updating the high average-utility patterns with pre-large concept," *Int. J. Speech Technol.*, vol. 50, no. 11, pp. 3788–3807, Nov. 2020.

[36] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, and W. Pedrycz, "Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases," *Future Gener. Comput. Syst.*, vol. 103, pp. 58–78, Feb. 2020.

[37] M. A. Fouad, W. Hussein, S. Rady, P. S. Yu, and T. F. Gharib, "An efficient approach for mining reliable high utility patterns," *IEEE Access*, vol. 10, pp. 1419–1431, 2022.

[38] T. Wei, B. Wang, Y. Zhang, K. Hu, Y. Yao, and H. Liu, "FCHUIM: Efficient frequent and closed high-utility itemsets mining," *IEEE Access*, vol. 8, pp. 109928–109939, 2020.

[39] T. D. D. Nguyen, L. T. T. Nguyen, L. Vu, B. Vo, and W. Pedrycz, "Efficient algorithms for mining closed high utility itemsets in dynamic profit databases," *Exp. Syst. Appl.*, vol. 186, Dec. 2021, Art. no. 115741.

[40] B. Vo, L. T. T. Nguyen, T. D. D. Nguyen, P. Fournier-Viger, and U. Yun, "A multi-core approach to efficiently mining high-utility itemsets in dynamic profit databases," *IEEE Access*, vol. 8, pp. 85890–85899, 2020.

[41] B. Vo, L. V. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, L. T. T. Nguyen, and T.-P. Hong, "Mining correlated high utility itemsets in one phase," *IEEE Access*, vol. 8, pp. 90465–90477, 2020.

[42] R. S. Almoqbily, A. Rauf, and F. H. Quradaa, "A survey of correlated high utility pattern mining," *IEEE Access*, vol. 9, pp. 42786–42800, 2021.

[43] Y. Baek, U. Yun, H. Kim, H. Nam, H. Kim, J. C.-W. Lin, B. Vo, and W. Pedrycz, "RHUPS: Mining recent high utility patterns with sliding Window–based arrival time control over data streams," *ACM Trans. Intell. Syst. Technol.*, vol. 12, no. 2, pp. 1–27, Apr. 2021.

[44] H. Kim, T. Ryu, C. Lee, H. Kim, E. Yoon, B. Vo, J. C. W. Lin, and U. Yun, "EHMIN: Efficient approach of list based high-utility pattern mining with negative unit profits," *Exp. Syst. Appl.*, vol. 209, Dec. 2022, Art. no. 118214.

[45] H. Nam, U. Yun, B. Vo, T. Truong, Z.-H. Deng, and E. Yoon, "Efficient approach for damped window-based high utility pattern mining with list structure," *IEEE Access*, vol. 8, pp. 50958–50968, 2020.

[46] N. Vuong, B. Le, T. Truong, and D.-P. Nguyen, "Efficient algorithms for discovering high-utility patterns with strong frequency affinities," *Exp. Syst. Appl.*, vol. 169, May 2021, Art. no. 114464.

[47] N. T. Tung, L. T. T. Nguyen, T. D. D. Nguyen, and B. Vo, "An efficient method for mining multi-level high utility itemsets," *Int. J. Speech Technol.*, vol. 52, no. 5, pp. 5475–5496, Mar. 2022.

[48] P. Fournier-Viger, W. Gan, Y. Wu, M. Nouioua, W. Song, T. C. Truong, and H. V. Duong, "Pattern mining: Current challenges and opportunities," in *Proc. Int. Conf. Database Syst. Advanced Appl.*, 2022, pp. 34–49.

[49] S. Datta, K. Mali, and U. Ghosh, "High occupancy itemset mining with consideration of transaction occupancy," *Arabian J. Sci. Eng.*, vol. 47, no. 2, pp. 2061–2075, Feb. 2022.

[50] H. Kim, T. Ryu, C. Lee, H. Kim, T. Truong, P. Fournier-Viger, W. Pedrycz, and U. Yun, "Mining high occupancy patterns to analyze incremental data in intelligent systems," *ISA Trans.*, vol. 131, pp. 460–475, Dec. 2022.

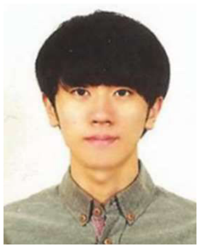
[51] G. M. Karthik, M. Sayeekumar, R. Kumaravel, and T. Aravind, "Finding spectrum occupancy pattern using CBFPP mining technique," *J. Intell. Fuzzy Syst.*, vol. 39, no. 3, pp. 4361–4368, Oct. 2020.

[52] J. Sun, W. Gan, J. C.-W. Lin, and H.-C. Chao, "Pattern discovery with utility occupancy," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2022, pp. 6261–6270.

[53] X. Dong, M. Wang, Y. Liu, G. Xiao, D. Huang, and G. Wang, "An efficient spatial high-utility occupancy frequent item mining algorithm for mission system integration architecture design using the MBSE method," *Aerosp. Syst.*, vol. 5, no. 3, pp. 377–392, Sep. 2022.

[54] C.-M. Chen, L. Chen, W. Gan, L. Qiu, and W. Ding, "Discovering high utility-occupancy patterns from uncertain data," *Inf. Sci.*, vol. 546, pp. 1208–1229, Feb. 2021.

[55] T. Ryu, U. Yun, C. Lee, J. C. Lin, and W. Pedrycz, "Occupancy-based utility pattern mining in dynamic environments of intelligent systems," *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 5477–5507, Sep. 2022.
[56] W. Gan, J. Lin, P. Fournier-Viger, H. Chao, J. Zhan, and J. Zhang, "Exploiting highly qualified pattern with frequency and weight occupancy," *Knowl. Inf. Syst.*, vol. 56, no. 1, pp. 165–196, Jul. 2018.
[57] Y. Baek, U. Yun, H. Kim, J. Kim, B. Vo, T. Truong, and Z.-H. Deng, "Approximate high utility itemset mining in noisy environments," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106596.
[58] J. Kim, U. Yun, E. Yoon, J. C.-W. Lin, and P. Fournier-Viger, "One scan based high average-utility pattern mining in static and dynamic databases," *Future Gener. Comput. Syst.*, vol. 111, pp. 143–158, Oct. 2020.
[59] A. Jamshed, B. Mallick, and P. Kumar, "Deep learning-based sequential pattern mining for progressive database," *Soft Comput.*, vol. 24, no. 22, pp. 17233–17246, Nov. 2020.
[60] J. Zhang, Y. Zhao, F. Shone, Z. Li, A. F. Frangi, S. Q. Xie, and Z.-Q. Zhang, "Physics-informed deep learning for musculoskeletal modeling: Predicting muscle forces and joint kinematics from surface EMG," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 31, pp. 484–493, 2023.



HYEONMO KIM received the B.S. degree in computer engineering from Sejong University, Seoul, South Korea, in 2022, where he is currently pursuing the M.S. degree. His research interests include data mining, information retrieval, database systems, and artificial intelligence.

TAEWOONG RYU received the B.S. degree in computer engineering from Sejong University, Seoul, South Korea, in 2022, where he is currently pursuing the M.S. degree. His research interests include data mining, information retrieval, database systems, and artificial intelligence.

CHANHEE LEE received the B.S. degree in computer engineering from Sejong University, Seoul, South Korea, in 2021, where he is currently pursuing the M.S. degree. His research interests include data mining, information retrieval, database systems, and artificial intelligence.

SINYOUNG KIM received the B.S. degree in computer engineering from Sejong University, Seoul, South Korea, in 2023, where he is currently pursuing the M.S. degree. His research interests include data mining, information retrieval, database systems, and artificial intelligence.

BAY VO (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, in 2002, 2005, and 2011, respectively. He is currently an Associate Professor and the Dean of the Faculty of Information Technology, Ho Chi Minh University of Technology (HUTECH), Ho Chi Minh City. His research interests include association rules, classification, and mining in incremental database.

JERRY CHUN-WEI LIN (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, in 2010. He is currently a Full Professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 400 research articles in journals, and international conferences. His research interests include data mining, soft computing, and artificial intelligence/machine learning. He is the Guest Editor/an Associate Editor of several IEEE/ACM journals.

UNIL YUN received the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2005. He is currently a Full Professor with the Department of Computer Engineering, Sejong University, Seoul, South Korea. He has published more than 200 research articles in refereed journals, and international conferences. His research interests include data mining, information retrieval, database systems, artificial intelligence, and digital libraries. He is an Associate Editor (Editorial Board Member) of *Knowledge-Based Systems*, *PLoS ONE*, and *Electronics*.