Efficient Mining of Concise Patterns for Frequent High-Utility Occupancy Itemsets Using Extended Pruning Strategies

Tien Hoang[1,2,3], Lan Huynh[4], Hai Duong[*,3], Tin Truong[3]

[1] Department of Computer Science, Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam
[3] Department of Mathematics and Computer Science, Dalat University, Dalat, Vietnam
[4] Faculty of Information Technology, HCM University of Industry and Trade, Vietnam

Correspondence: Hai Duong. Email: haidv@dlu.edu.vn
Communication: received 01/03/2025, revised 14/04/2025, accepted 20/05/2025
DOI: 10.32913/mic-ict-research.v2025.n2.1360

***Abstract*: Concise representations of frequent high utility occupancy itemsets (FHUOIs), including maximal FHUOIs, closed FHUOIs, and generators of FHUOIs, are essential in utility-driven pattern mining. These representations offer several advantages over the complete set of FHUOIs, such as reduced size, improved efficiency, lower storage costs, and easier analysis. Notably, maximal FHUOIs allow for the recovery of all FHUOIs, while closed FHUOIs and generators enable the generation of non-redundant high utility occupancy rules and the efficient reconstruction of all FHUOIs along with their key information. Despite their significance, existing methods either mine closed FHUOIs and generators separately or lack a solution for mining all maximal FHUOIs. To bridge this gap, this paper introduces two novel algorithms *MaxFHUOI-Miner* and *CGFHUOI-Miner*. The former efficiently extracts only maximal FHUOIs using extended pruning strategies that eliminate non-maximal itemsets early, while the latter simultaneously mines closed FHUOIs and generators by employing innovative pruning techniques to eliminate non-closed and non-generator itemsets at three levels of the prefix tree without performing subset checks. Extensive experiments on real-world and synthetic datasets demonstrate that the proposed algorithms outperform existing and baseline methods in both speed and memory efficiency, particularly for low minimum support and utility occupancy thresholds in dense databases.**

Keywords: *Frequent high utility occupancy itemset; Pruning strategy; Weak upper bound; Weak lower bound, Concise representations*

I. INTRODUCTION

Frequent High Utility Occupancy Itemset (FHUOI) mining (FHUOIM) is a fundamental task in data mining that focuses on identifying valuable patterns in quantitative transaction databases (QTDBs). It aims to discover itemsets that not only occur frequently but also contribute significantly to the total utility of transactions [1][2][3][4]. In FHUOIM, the utility occupancy of an itemset A in a transaction T measures its contribution to T 's overall utility. The total utility occupancy of A is calculated as the average occupancy across all transactions in which it appears. An itemset qualifies as an FHUOI if both its average utility occupancy and support exceed predefined thresholds. FHUOIM is designed to efficiently extract all such FHUOIs from a QTDB. This technique has various real-world applications, such as improving recommendation systems based on mobile application traffic patterns and analyzing web log data for user behavior insights [5][6].

A main challenge in FHUOIM is the large number of patterns generated, especially in dense QTDBs or when the support and utility occupancy thresholds are low. This leads to longer execution times, higher memory usage, and makes it difficult for users to interpret and analyze the results. A more effective solution is to focus on concise representations of FHUOIs, such as closed FHUOIs (CFHUOIs) [7], maximal FHUOIs (MFHUOIs), and generators of FHUOIs (GFHUOIs) [8][9], rather than extracting all FHUOIs. These three representations help reduce redundancy and improve efficiency. An MFHUOI is an FHUOI that is not strictly contained in any other FHUOI. The $\mathcal{MFHUOI}$ set of all MFHUOIs is usually much smaller than the full set $\mathcal{FHUOI}$ of all FHUOIs, making it more manageable for users. Additionally, MFHUOIs can be used to reconstruct the complete set of FHUOIs. A CFHUOI is an FHUOI that has no larger FHUOI with the same support, while a GFHUOI is an FHUOI with no smaller FHUOI having the same support. The $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets of CFHUOIs and GFHUOIs are generally smaller than

$\mathcal{FHUOI}$ but larger than the $\mathcal{MFHUOI}$ set. Notably, CFHUOIs and GFHUOIs represent the largest and smallest elements, respectively, within each equivalence class. In this context, an equivalence class is defined as a group of itemsets that share identical support and occur in the same set of transactions in the database [10].

In this paper, we focus on mining the $\mathcal{MFHUOI}$, $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets for several key reasons. First, these sets share a major advantage: they are significantly smaller than the $\mathcal{FHUOI}$ set of all FHUOIs. This allows for more efficient discovery, reduced storage requirements, and easier interpretation for users. Second, each set offers distinct benefits. The $\mathcal{MFHUOI}$ set is the smallest among the three and can be used to reconstruct all FHUOIs. Additionally, MFHUOIs are valuable for identifying frequent longest common patterns in real-world applications. The $\mathcal{CFHUOI}$ set, while larger than $\mathcal{MFHUOI}$, is lossless, meaning it retains complete frequency information for all itemsets. CFHUOIs are useful because they represent the largest itemsets with high utility occupancy in specific transaction groups (such as customer groups). They are the most representative patterns in their equivalence classes, itemsets that appear in the same transactions. This makes CFHUOIs highly applicable in practical scenarios. For the $\mathcal{GFHUOI}$ set, researchers argue that it aligns with the minimum description length principle [10], making it preferable because GFHUOIs are the smallest representatives of their equivalence classes. Finally, CFHUOIs and GFHUOIs can be combined to form non-redundant high utility occupancy association rules, where a GFHUOI appears on the left-hand side and a CFHUOI on the right-hand side. They also enable fast reconstruction of all FHUOIs in equivalence classes, along with their support values.

The above advantages emphasize the importance of mining three sets $\mathcal{MFHUOI}$, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$ and drive the need for optimizing existing algorithms to extract these sets more efficiently.

Contributions. As discussed earlier, each of the three condensed representations of FHUOIs serves distinct purposes depending on the mining objective. In many scenarios, the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets are extracted together to generate non-redundant high utility occupancy rules and efficiently recover all FHUOIs, along with their supports and utility occupancies. In contrast, $\mathcal{MFHUOI}$, being the smallest set, is often mined separately for specific applications. Despite their importance, to our knowledge, no existing algorithm can simultaneously mine both $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$, or independently extract $\mathcal{MFHUOI}$. To bridge this gap, this paper introduces two novel algorithms: one for concurrently discovering both $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$, and another dedicated to efficiently mining $\mathcal{MFHUOI}$ alone.

The main contributions of this paper are as follows. First, we introduce two extended pruning strategies, EPNonMaxBr and LPNonMaxBr, designed to eliminate unpromising branches early in the prefix tree that cannot contain any MFHUOIs. These strategies are integrated into a novel algorithm, MaxFHUOI-Miner, to efficiently extract the $\mathcal{MFHUOI}$ set. Second, we propose two pruning techniques, EPNonCGBr and LPNonCGBr, that reduce the search space by eliminating branches that cannot contain both CFHUOIs and GFHUOIs. Using these techniques, we develop CGFHUOI-Miner, a novel algorithm that discovers both CFHUOIs and GFHUOIs within a single process. To the best of our knowledge, CGFHUOI-Miner and MaxFHUOI-Miner are the first algorithms specifically designed for mining CFHUOIs and GFHUOIs together and MFHUOIs independently, respectively. Finally, we provide experimental evidence demonstrating that the proposed algorithms outperform existing and baseline methods in terms of runtime, memory efficiency, and scalability.

The rest of this paper is structured as follows. Section 2 reviews related work, followed by Section 3, which introduces fundamental concepts and notation for identifying FHUOIs and their concise representations. Section 4 details the core theoretical contributions, including innovative pruning strategies to efficiently filter out non-closed, non-generator, and non-maximal FHUOIs, along with the introduction of two novel algorithms, MaxFHUOI-Miner and CGFHUOI-Miner. Section 5 presents the experimental analysis of these algorithms. Finally, Section 6 summarizes the findings and discusses potential directions for future research.

II. RELATED WORK

1. Mining high utility itemsets and high utility occupancy itemsets

The problem of high utility itemset (HUI) mining (HUIM) in QTDBs [11] has been a major focus of research in recent years. A significant challenge in HUIM is that the utility function u does not exhibit the anti-monotonic (or Downward Closure) property, which is a key principle for efficiently reducing the search space in pattern mining. To tackle this limitation, researchers have introduced upper bounds on u, including the anti-monotonic Transaction Weighted Utility (TWU) [12] and a more refined upper bound that accounts for the remaining utility of a pattern in its supporting transactions. This enhanced bound is implemented in the HUI-Miner algorithm, which is built on a utility-list-based framework [13]. To address the

HUIM problem, various algorithms have been developed. UP-Growth [14] employs a utility pattern tree (UP-Tree) while requiring only two scans of the database. EFIM [15] enhances mining efficiency by incorporating two upper bounds: sub-tree utility and local utility. HMiner [16] utilizes a compact utility list along with a virtual hyperlink data structure to efficiently manage itemset information. Meanwhile, Hamm [17] is a single-phase algorithm that introduces a novel data structure, TV, which combines a prefix tree with a utility vector for improved performance.

High Utility Itemset Mining (HUIM) has been widely applied in various domains; however, the conventional utility measure u of an itemset A in a supporting transaction T (also referred to as the absolute utility of A in T [18]) does not adequately capture its utility occupancy level within T. Utility occupancy quantifies the relative contribution of A to the total transaction utility, providing a more granular assessment of its significance for each transaction. This improved measure is key to accurately finding important patterns in real-world applications, like personalized app and webpage recommendations [5]. To address this limitation, Shen et al. [5] introduced the concept of utility occupancy in QTDBs to extend occupancy-based analysis. The utility occupancy of an itemset A in a supporting transaction T is defined as the ratio of its utility in T to the total utility of T, representing the relative utility of A in T. This measure highlights the importance of A in terms of utility distribution across transactions. The overall utility occupancy of A in a QTDB, denoted as $occ(A)$, is computed as the average of its utility occupancies across all transactions containing A. An itemset is considered a Frequent High Utility Occupancy Itemset (FHUOI) if its support and utility occupancy meet or exceed the user-defined min_sup and min_uo thresholds, respectively. This constraint ensures that extracted patterns are not only frequent but also highly relevant in terms of utility contribution, making them valuable for various analytical and decision-making tasks in data mining.

The problem of Frequent High Utility Occupancy Itemset Mining (FHUOIM) aims to identify all FHUOIs in a QTDB. This task has been widely applied in various domains, particularly in recommendation systems. For example, in mobile app traffic analysis, each mobile application is treated as an item, and its data usage represents its utility. By mining FHUOIs, analysts can address critical questions such as: "Which essential mobile applications contribute the most to users' overall data consumption?" [5]. Traditional pattern mining methods, which focus solely on high utility or high frequency, are insufficient for answering such queries, as they do not account for the relative importance of an item's utility in each transaction. Another key application is webpage recommendation based on web browsing logs, where each website is considered as an item, and its browsing duration serves as its utility. Identifying FHUOIs provides deeper insights into user engagement, enabling more tailored content recommendations and enhanced user experiences. By capturing both utility and occupancy, FHUOIM provides a more comprehensive and informative approach to pattern discovery in transactional datasets.

A key challenge in FHUOIM is that the occupancy measure (occ), like utility, lacks anti-monotonicity, making it difficult to efficiently prune the search space. To tackle this problem, the authors in [5] introduced an upper bound (UB) on occ and proposed the OCEAN algorithm to facilitate FHUOIM by reducing unnecessary computations. To further enhance the efficiency of FHUOIM algorithms, Gan et al. [19] developed the HUOPM algorithm, which efficiently discovers the complete set of FHUOIs. This algorithm efficiently removes infrequent or low utility occupancy itemsets by using the anti-monotonicity of support and an upper bound on occ. More recently, He et al. [6] introduced SHO, a method specifically designed for handling large-scale datasets. By employing a suffix-based partitioning technique, SHO significantly improves computational efficiency, allowing for faster and more scalable FHUOIM. Beyond these developments, researchers have expanded FHUOIM to new problem settings. For instance, some studies have focused on mining potential FHUOIs in uncertain QTDBs [20], while others have explored frequent high utility occupancy sequential pattern mining in quantitative sequence databases [21]. These extensions broaden the applicability of FHUOIM, making it more suitable for uncertain and sequential data environments in real-world applications.

2. Mining concise representations for HUIs and FHUOIs

High utility itemsets (HUIs) are important in many real-world applications, but their cardinality can be excessively large, especially in large, dense QTDBs or when the minimum utility threshold is too low. This makes analyzing and utilizing HUIs both difficult and time-consuming. To address this challenge, researchers have developed algorithms that generate compact representations of HUIs, significantly reducing the result set while retaining essential information. Notable methods for this include CHUI-Miner(Max) [22] and MaxC-HUIM [23] for mining maximal HUIs; CHUD [18], CHUI-Miner [24], EFIM-Closed [25], HMiner-Closed [26], and C-HUIM [23] for exploiting closed HUIs; and approaches introduced in [27] for high utility generators. A high utility pattern A is maximal if no proper super-

itemset of A is an HUI. It is closed (or a generator) if no proper HUI super-itemset (or sub-itemset, respectively) of A has the same support. Note that maximal HUIs provide a compact representation of HUIs, with their cardinality being smaller than those of closed HUIs and HUI generators. In contrast, closed and generator HUIs not only offer concise representations but also retain all essential information, ensuring a lossless representation of HUIs [18]. Notably, within each equivalence class of HUIs that share the same supporting transactions, there exists a unique closed HUI, which represents the largest pattern in the class under the traditional set inclusion relation ($\subseteq$). However, multiple HUI generators may exist in the same class, each corresponding to a minimal pattern [18][27].

The challenge of managing FHUOIs is similar, as their cardinality often becomes excessively large, making both analysis and practical application difficult. This underscores the necessity of discovering concise representations for FHUOIs. To address this issue, the CloFHUOIM algorithm [7] was introduced as the first efficient method for extracting compact representations of FHUOIs by identifying the $\mathcal{CFHUOI}$ set, which consists of all closed FHUOIs (CFHUOIs). Furthermore, the authors in [7] proposed the MaxCloFHUOIM algorithm, which simultaneously discovers both CFHUOIs and maximal FHUOIs (MFHUOIs) in a single process. To enhance efficiency, a novel weak upper bound, $wubocc$, was introduced for the utility occupancy function. This bound is tighter than the previous upper bound $\widehat{\Phi}$ [19] and is used in an effective pruning strategy to reduce the search space. Additionally, two algorithms have been developed for mining the $\mathcal{GFHUOI}$ set of all generators of FHUOIs (GFHUOIs): GFHUOI-Miner and Gen-FHUOIM [8][9]. Since the occ function lacks the monotonicity property in each equivalence class, the authors devised weak lower bounds on occ and corresponding pruning techniques to efficiently eliminate non-generator FHUOIs early in the mining process. Experimental results confirm that these algorithms significantly outperform baseline approaches in terms of runtime, memory consumption, and scalability on large datasets.

III. PRELIMINARIES AND PROBLEM DEFINITION

This section presents concepts and notation related to the task of mining frequent high utility occupancy itemsets (FHUOIs) and their concise representations in QTDBs. The input data format for this task is formally defined as follows.

Definition 1 *(Quantitative transaction database).* Let $\mathcal{A} \overset{\text{def}}{=} \{a_j, j \in J \overset{\text{def}}{=} \{1, 2, ..., N'\}\}$ represent a finite set of distinct items, and let $X \subseteq \mathcal{A}$ denote a subset of $\mathcal{A}$. If X contains exactly k items, it is referred to as a k-itemset, where $k = |X|$. Without loss of generality, we assume that the items in each itemset are ordered according to a total order $\prec$ (e.g., lexicographic order). The profit vector $\mathcal{P} \overset{\text{def}}{=} \left(p\left(a_j\right), j \in J \text{ and } p\left(a_j\right) > 0\right)$ represents the external utility of all items in the set $\mathcal{A}$, where $p\left(a_j\right)$ denotes the relative importance of each item a_j. This importance can be quantified by attributes such as unit profit, price, weight, or other relevant factors. A q-item is defined as a pair (a, q), where a is an item from $\mathcal{A}$ and q is a positive integer representing the purchase quantity or internal utility of item a. A quantitative database (QTDB) $\mathcal{D}$ is defined as a collection of transactions, $\mathcal{D} \overset{\text{def}}{=} \left\{T_i, i \in I \overset{\text{def}}{=} \{1, 2, \ldots, N\}\right\}$, where each transaction T_i is represented as $T_i \overset{\text{def}}{=} \left\{\left(a_j, q_{ij}\right), j \in J_i\right\}$, consisting of a set of q-items. Each transaction T_i is uniquely identified by a transaction identifier (TID), with TID = i and J_i is a subset of J, $J_i \subseteq J$. The utility of a q-item (a, q), denoted as $u((a, q))$, is defined as:

$$u((a, q)) \overset{\text{def}}{=} q \times p(a) \tag{1}$$

where $p(a)$ is the external utility of item a. Intuitively, this utility represents the profit generated by the sale of item a within the transaction where the q-item (a, q) apprears.

Definition 2 *(Integrated QTDB).* To eliminate redundant computations of the utility u for q-items $\left(a_j, q_{ij}\right)$ across transactions T_i in $\mathcal{D}$, these utility values are precomputed and stored in an integrated quantitative database, referred to as $\mathcal{D}'$, which is defined as:

$$\mathcal{D}' \overset{\text{def}}{=} \left\{T_i' \overset{\text{def}}{=} \left\{\left(a_j, q_{ij}'\right), j \in J_i\right\}, J_i \subseteq J, i \in I\right\} \tag{2}$$

where $q_{ij}' = q_{ij} * p\left(a_j\right)$.

Table I
A QTDB $\mathcal{D}$

TID	Transaction (item, quantity)
T_1	$(a, 1), (e, 8)$
T_2	$(b, 1), (c, 12), (d, 12), (e, 21), (f, 1)$
T_3	$(a, 2)$
T_4	$(a, 1), (b, 5), (c, 2), (d, 4), (e, 26)$
T_5	$(a, 3), (e, 9), (f, 1)$

Table II
A PROFIT VECTOR $\mathcal{P}$

a_j	a	b	c	d	e	f
$p\left(a_j\right)$	16	2	1	3	2	55

Running Example: Table 1 and Table 2 present a QTDB $\mathcal{D}$ and an external utility table $\mathcal{P}$, respectively. Table 3 illustrates the integrated QTDB $\mathcal{D}'$, obtained by combining $\mathcal{D}$ and $\mathcal{P}$. This integrated database, $\mathcal{D}'$, serves as a running example throughout the study to illustrate key concepts.

Table III
THE INTEGRATED QTDB $\mathcal{D}'$

Tid	Quantitative Transaction	$tu(T_i)$
T_1	$(a,16),(e,16)$	32
T_2	$(b,2),(c,12),(d,36),(e,42),(f,55)$	147
T_3	$(a,32)$	32
T_4	$(a,16),(b,10),(c,2),(d,12),(e,52)$	92
T_5	$(a,48),(e,18),(f,55)$	121
	$TU=$	424

For clarity and brevity, each transaction T_i' in $\mathcal{D}'$ will be referred to as T_i, and the shorthand ae will represent the itemset $\{a,e\}$.

Definition 3 *(Support of an Itemset).* For any $X \subseteq \mathcal{A}$, where $X \neq \varnothing$, let $\rho(X)$ represent the set of transactions in $\mathcal{D}'$ that contain X.

$$\rho(X) \stackrel{\text{def}}{=} \left\{T_i \in \mathcal{D}' \mid q'_{ij} > 0, \forall a_j \in X\right\} \tag{3}$$

By convention, $\rho(\varnothing) \stackrel{\text{def}}{=} \mathcal{D}'$. The support of X, denoted as $supp(X)$, is defined as the number of transactions in the QTDB $\mathcal{D}'$ that contain X. Formally, $supp(X)$ is the cardinality of the set $\rho(X)$. Only non-empty itemsets X with at least one occurrence in the transactions of $\mathcal{D}'$ are considered, i.e., $X \in \mathcal{IS} \stackrel{\text{def}}{=} \{X \subseteq \mathcal{A} \mid X \neq \emptyset \wedge \rho(X) \neq \varnothing\}$.

Example 1. For itemset $X = ae$, as $X \subseteq T_1, X \subseteq T_4$ and $X \subseteq T_5$, we have $\rho(X) = \{T_1, T_4, T_5\}$ and $supp(X) = 3$.

Definition 4 *(Itemset utility in a transaction and utility function).* The utility of an item a_j (where $j \in J$) in a transaction $T_i \in \mathcal{D}'$ is denoted and defined as $u\left(a_j, T_i\right) \stackrel{\text{def}}{=} q'_{ij}$. For an itemset X in T_i, where T_i contains X, the utility of X in T_i is denoted as $u(X,T_i) \stackrel{\text{def}}{=} \sum_{a_j \in X} u\left(a_j, T_i\right)$. The utility function u of an itemset X across $\mathcal{D}'$ is denoted by $u(X)$ and defined as: $u(X)$ and $u(X) \in R_+ \stackrel{\text{def}}{=} \{x \in \mathbb{R} \mid x > 0\}$.

$$u(X) \stackrel{\text{def}}{=} \sum_{T_i \in \rho(X)} u(X,T_i) \tag{4}$$

where $u(X,T_i) \stackrel{\text{def}}{=} \sum_{a_j \in X} u\left(a_j, T_i\right)$

Definition 5 *(Transaction utility and total utility).* The utility of a transaction T_i is denoted $tu(T_i)$ and is defined by $tu(T_i) \stackrel{\text{def}}{=} \sum_{\left(a_j, q'_{ij}\right) \in T_i} q'_{ij}$. The total utility (TU) of $\mathcal{D}'$ is the sum of the utilities of all transactions in $\mathcal{D}'$, expressed as

$$TU \stackrel{\text{def}}{=} \sum_{T_i \in \mathcal{D}} tu(T_i) \tag{5}$$

where $tu(T_i) \stackrel{\text{def}}{=} \sum_{\left(a_j, q'_{ij}\right) \in T_i} q'_{ij}$.

Example 2. For itemset $X = ae$, then $\rho(X) = \{T_1, T_4, T_5\}, u(X,T_1) = 16+16 = 32$, $u(X,T_4) = 16+52 = 68$, and $u(X,T_5) = 48+18 = 66$. Then, $u(X) = u(X,T_1) + u(X,T_4) + u(X,T_5) = 32+68+66 = 166$. We have $tu(T_1) = 16+16 = 32$ and $TU = 32+147+32+92+121 = 424$.

Definition 6 *(Utility Occupancy).* The utility occupancy of an itemset $X \in \mathcal{A}$ in $\mathcal{D}'$ is denoted as $occ(X)$ and defined as follows:

$$occ(X) \stackrel{\text{def}}{=} \frac{\sum_{T_i \in \rho(X)} u_{rel}(X,T_i)}{supp(X)}, \tag{6}$$

where $u_{rel}(X,T_i) \stackrel{\text{def}}{=} \frac{u(X,T_i)}{tu(T_i)}$ represents the utility occupancy ratio of X relative to the utility of the input transaction T_i.

Definition 7 *(Frequent High Utility Occupancy Itemset).* Given two user-defined positive thresholds, minimum utility occupancy $min\_uo \in (0,1]$ and minimum support min_sup, an itemset X is called a high utility occupancy itemset (HUOI) if its utility occupancy in $\mathcal{D}'$ satisfies the condition $occ(X) \geq min\_uo$. Conversely, if $occ(X) < min\_uo$, then X is termed a low utility occupancy itemset (LUOI). Additionally, an HUOI X is further classified as a frequent high utility occupancy itemset (FHUOI) if its support meets or exceeds the threshold min_sup, i.e., $supp(X) \geq min\_sup$.

The problem of FHUOI mining ($\mathcal{FHUOIM}$) is to identify the set of all FHUOIs, defined as

$$\mathcal{FHUOI} \stackrel{\text{def}}{=} \{X \in \mathcal{A} \mid occ(X) \geq min\_uo \wedge supp(X) \geq min\_sup\} \tag{7}$$

Equivalently, this set can be expressed as $\mathcal{FHUOI} = \mathcal{HUOI} \cap \mathcal{FI}$, where $\mathcal{HUOI} \stackrel{\text{def}}{=} \{X \in \mathcal{A} \mid occ(X) \geq min\_uo\}$ and $\mathcal{FI} \stackrel{\text{def}}{=} \{X \in \mathcal{A} \mid supp(X) \geq min\_sup\}$ are the sets of all HUOIs and frequent itemsets (FIs), respectively. It is important to note that $\mathcal{FHUOIM}$ generalizes the traditional HUOI mining problem, where $min\_sup = 1$ represents a special case.

Example 3. For $min\_sup = 2$ and $min\_uo = 0.7$, consider the itemset $X = ae$. Then, $\rho(X) = \{T_1, T_4, T_5\}, supp(X) = 3$, we have $u_{rel}(X,T_1) = (16+16)/32 = 1$, $u_{rel}(X,T_4) = (16+52)/92 = 0.739$ and $u_{rel}(X,T_5) = (48+18)/121 = 0.545$. The utility occupancy of X in $\mathcal{D}'$ is $occ(X) = (1+0.739+0.545)/3 = 0.762 \geq min\_uo$ and $supp(X) \geq min\_sup$. Thus, X is an FHUOI. Meanwhile, the itemset $Y = c$ is a low utility occupancy itemset because $occ(Y) = 0.0517 < min\_uo$.

We assume that all items in $\mathcal{A}$ are ordered according to a total order relation $\prec$. For any item y and itemsets X and Y, we denote $X \prec y$ if and only if (or briefly, iff) $x \prec y, \forall x \in X$, or $X \prec Y$ iff $x \prec y, \forall x \in X, \forall y \in Y$. If $X \cap Y = \emptyset$, the union of the two disjoint itemsets X and Y is denoted as $X + Y$. Furthermore, if $X \cap Y = \emptyset$ and $X \prec Y$, the notation X + Y is replaced by $X \oplus Y$ to indicate the ordered union of the two disjoint itemsets.

Definition 8 *(Forward and Backward Extensions of an Itemset)*. For any two itemsets X and Y where $X \cap Y = \emptyset$ and $X \prec Y$, the itemset $S = X \oplus Y$ is referred to as a forward extension (FE), or simply an extension, of X with Y. If $Y \neq \emptyset$, then S is called a proper FE of X. Given an itemset $Y = X \oplus y$ with $X \prec y$, a new itemset $B = (X + P) \oplus y$, where $P \prec y$, is formed by inserting the items of P into X before y, in accordance with the order relation $\prec$. This itemset B is referred to as a backward extension of Y.

Specifically, given an itemset $Y = X \oplus y$, where Y is formed by appending an item y to X, if $supp(Y) = supp(X)$, then Y is defined as a $1-$ forward of X (with y). Similarly, if $supp(C) = supp(Y)$ for $Y = X \oplus y \subset C = (X + x) \oplus y$ with $x \notin X, X \prec y$ and $x \prec y$, meaning C is derived from Y by inserting an item x before the last item y of Y, then C is referred to as a $1-$ backward of Y (with x). It is evident that a $1-$ forward of X also constitutes a proper forward extension of X.

The operator $\oplus$ plays a crucial role in high utility occupancy itemset mining (HUOIM) algorithms, as it enables the systematic exploration of larger itemsets in the search space.

Example 4. For instance, consider $Y = bd$. Then, $bde = Y \oplus e$ is a $1-$ forward of Y with e because $supp(Y) = supp(bde) = 3$. However, bdf is not a $1-$ forward of bd since $supp(bdf) = 1 \neq supp(bd) = 2$, although bdf is an FE of bd. Similarly, the itemset bcd is also a $1-backward$ of Y (with c) as $supp(Y) = supp(bcd) = 2$. In contrast, abd is not a $1-backward$ of Y(with a) since $supp(Y) = 2 \neq supp(abd) = 1$.

Definition 9 (*Set $\mathcal{CFHUOI}$*). Let X be an FHUOI. The itemset X is termed a closed frequent high utility occupancy itemset (CFHUOI) if no FHUOI exists as a proper superitemset of X with the same support. The set of all CFHUOIs is denoted and defined as:

$$\mathcal{CFHUOI} \stackrel{\text{def}}{=} \{X \in \mathcal{FHUOI} \mid \nexists Y \in \mathcal{FHUOI} : Y \supset X \wedge supp(Y) = supp(X)\} \tag{8}$$

Definition 10 (*Set $\mathcal{GFHUOI}$*). Let X be an FHUOI, i.e., $X \in \mathcal{FHUOI}$. X is referred to as a generator of frequent high utility occupancy itemset (GFHUOI) if no FHUOI exists as a proper sub-itemset of X with the same support. The collection of all such GFHUOIs is denoted as $\mathcal{GFHUOI}$, which is formally defined as:

$$\mathcal{GFHUOI} \stackrel{\text{def}}{=} \{X \in \mathcal{FHUOI} \mid \nexists Y \in \mathcal{FHUOI} : Y \subset X \wedge supp(Y) = supp(X)\} \tag{9}$$

Definition 11 (*Set $\mathcal{MFHUOI}$*). An FHUOI X is said to be *a maximal frequent high utility occupancy itemset* (MFHUOI) if *no FHUOI exists as a proper* superset of X. The set of all MFHUOIs is denoted and defined as

$$\mathcal{MFHUOI} \stackrel{\text{def}}{=} \{X \in \mathcal{FHUOI} \mid \nexists Y \in \mathcal{FHUOI} : Y \supset X\} \tag{10}$$

Problem statement. Given a quantitative transaction database (QTDB) $\mathcal{D}'$ and two userdefined thresholds ($min\_uo \in (0; 1]$ for minimum utility occupancy and min_sup for minimum support), this study develops two efficient algorithms: (1) for simultaneously mining both $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets, and (2) for discovering the $\mathcal{MFHUOI}$ set separately.

Example 5. Consider $min\_uo = 0.4$, $min\_sup = 2$, and $X = ef$. We have $occ(X) = 0.63 \geq min\_uo$ and $supp(X) = 2 \geq min\_sup$. Since no FHUOI exists as a superset of X with the same support, X qualifies as a CFHUOI. However, X is not a GFHUOI, because there exists an FHUOI $Y = f \in \mathcal{FHUOI}$ with $occ(Y) = 0.41$, $X \supset Y$, and $supp(X) = supp(Y) = 2$. Y is classified as a GFHUOI since $\nexists Z \in \mathcal{FHUOI} : Y \supset Z$ and $supp(Z) = supp(Y)$. In addition, $Z = bcde$ is identified as an MFHUOI as $occ(Z) = 0.73, supp(Z) = 2$, and $\nexists Y \in \mathcal{FHUOI} : Y \supset Z$. Similarly, we have $\mathcal{GFHUOI} = \{a, ae, f, bd, be, ce, de\}$, $\mathcal{CFHUOI} = \{a, ae, ef, bcde\}$ and $\mathcal{MFHUOI} = \{ae, ef, bcde\}$. For $min\_uo = 0.1$ and $min\_sup = 1$, then $|\mathcal{GFHUOI}| = 15$, $|\mathcal{CFHUOI}| = 8$, and $|\mathcal{MFHUOI}| = 3$, while $|\mathcal{FHUOI}| = 47$.

For clarity, Table 4 summarizes the key abbreviations and notations used throughout this paper.

Table IV
ABBREVIATIONS AND NOTATIONS

Abbreviation	Explanation
HUOI	A high utility occupancy itemset
FI	A frequent itemset A, supp$(A) \geq min_sup$
FHUOI	A frequent high utility occupancy itemset ($occ(A) \geq min\_uo$ and $supp(A) \geq min\_sup$)
$\mathcal{HUOI}$	The set of all HUOIs
$\mathcal{FI}$	The set of all FIs
$\mathcal{FHUOI}$	The set of all FHUOIs, $\mathcal{FHUOI} = \mathcal{HUOI} \cap \mathcal{FI}$
CFHUOI	A closed frequent high utility occupancy itemset
GFHUOI	A generator of frequent high utility occupancy itemset
MFHUOI	A maximal frequent high utility occupancy itemset
$\mathcal{CFHUOI}$	The set of all CFHUOIs
$\mathcal{GFHUOI}$	The set of all GFHUOIs
$\mathcal{MFHUOI}$	The set of all MFHUOIs
$wubocc$	A weak upper bound (WUB) on occ
$wlbocc$	A novel weak lower bound (WLB) on occ
$NonC\_FHUO$	A non-closed frequent high utility occupancy
$NonG\_FHUO$	A non-generator frequent high utility occupancy
$NonCG\_FHUO$	A non-closed and non-generator frequent high utility occupancy

Consider the process of searching for candidate itemsets to identify the set of $\mathcal{FHUOI}$ or its concise representa-

tions, such as $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$, performed on a prefix search tree (or briefly, prefix tree). In this context, a prefix tree is a data structure that helps organize itemsets by grouping those with the same prefixes [28]. The tree starts from an empty itemset at the root, and each node represents one item. The path from the root to any node forms an itemset. We explore the tree using depth-first search (DFS), following a predefined item order $\prec$. For simplicity, in this paper, each node P stands for a complete itemset, and each child of P represents one of its forward extensions (see Figure 1). We denote the branch $Br(X)$ as the set containing X and all its proper FEs, and the proper branch $PBr(X)$ as $Br(X)$ excluding the itemset X itself.

IV. MINING CONCISE REPRESENTATIONS OF FHUOIS

Note that the goal of this paper is to discover the $\mathcal{CFHUOI}, \mathcal{GFHUOI}$ and $\mathcal{MFHUOI}$ sets. To enhance mining efficiency, it is crucial to prune unpromising branches $Br(A)$ or $PBr(A)$) from the prefix tree rooted at itemset A as early as possible. They include: (1) infrequent itemsets, (2) low utility occupancy itemsets (LUOIs), and (3) non-closed, non-generator, or non-maximal FHUOIs, i.e., FHUOIs that are neither closed, generators, nor maximal patterns. Infrequent itemsets can be eliminated early using the anti-monotonicity property of support: if an itemset A is infrequent, all its supersets are also infrequent. Thus, when A is infrequent, its entire branch $Br(A)$ can be pruned immediately. However, frequent itemsets can still be numerous, especially in large QTDBs with low min_sup values. To address this, the next sections introduce efficient pruning strategies to quickly discard LUOIs and non-closed, non-generator, or non-maximal FHUOIs.

1. Pruning low utility occupancy itemsets

Since the function occ is not anti-monotonic ($\mathcal{AM}$), the mining process requires upper bounds (UBs) or weak UBs (WUBs) on occ that exhibit $\mathcal{AM}$ or $\mathcal{AM}$-like properties [29] to facilitate the early elimination of LUOIs. A function ub is a UB on occ if it satisfies $occ(C) \leq ub(C), \forall C \in \mathcal{IS}$. Similarly, a function wub is a WUB on occ if $occ(S) \leq wub(C)$ for any itemset C and its proper FE S, where $S = C \oplus D \supset C$ with $D \neq \emptyset$. Among two UBs, ub_1 is considered tighter than ub_2 if $ub_1(C) \leq ub_2(C), \forall C \in \mathcal{IS}$.

Two existing (W)UBs on occ have been introduced for FHUOIM: the UB $\widehat{\Phi}$ [19] and the WUB $wubocc$ [7]. Since $wubocc$ is proven to be tighter than $\widehat{\Phi}$, this study employs $wubocc$ for early LUOI elimination based on the EPLUOI pruning strategy [7]. The definition and properties of $wubocc$ are presented below.

Let us denote $h(A) \stackrel{\text{def}}{=} \{T_i \in \mathcal{D}' \mid T_i \supseteq A \oplus E, E \neq \emptyset\}$ as the set of all transactions that contain at least a proper FE of A, so $h(A)$ is a subset of $\rho(A)$.

Definition 12 (*WUB wubocc on occ* [7]). For any itemset A and transaction T_i such that $|h(A)| \geq min\_sup$ and $T_i \in h(A)$, let us denote $\text{rem}(A, T_i)$ as an itemset that contains all remaining items in T_i after A. Let define two functions $ub_r(A, T_i) \stackrel{\text{def}}{=} u(A, T_i) + u(\text{rem}(A, T_i))$ and $ub_{r\_rel}(A, T_i) \stackrel{\text{def}}{=} \frac{ub_r(A,T_i)}{tu(T_i)}$. After sorting the series $\{ub_{r\_rel}(A, T_i) \mid T_i \in h(A)\}$ in descending order to create the new series $\{wubocc_i, i = 1..n\}$, the WUB $wubocc$ on occ is defined as follows:

$$wubocc(A) \stackrel{\text{def}}{=} \frac{1}{min\_sup} \sum_{1 \leq i \leq min\_sup} wubocc_i. \quad (11)$$

Proposition 1 (*The property of wubocc* [7]). $wubocc$ is a WUB on occ, i.e. $occ(B) \leq wubocc(A)$ for any itemset $B = A \oplus E$ being the proper FE of A with $E \neq \emptyset$.

Pruning strategy EPLUOI (*Early Pruning of LUOIs*). If $wubocc(A) < min\_uo$, the search for FHUOIs in the proper branch $PBr(A)$ is terminated, as Proposition 1 guarantees that no FHUOI exists in this proper branch.

Example 6. For $min\_sup = 1$ and $min\_uo = 0.93$, consider the itemset $X = bd$. Since $\rho(X) = h(X) = \{T_2, T_4\}$, then $ub_{r\_rel}(X, T_2) = \frac{2+36+42+55}{147} = 0.918$ and $ub_{r\_rel}(X, T_4) = \frac{10+12+52}{92} = 0.804$. After sorting the series $\{0.918, 0.804\}$ in descending order to create the new series $\{wubocc_i, i = 1..2\} = \{0.918, 0.804\}$, we have $wubocc(A) \stackrel{\text{def}}{=} \frac{1}{min\_sup} \sum_{1 \leq i \leq min\_sup} wubocc_i = wubocc_1 = 0.918 < min\_uo$. Thus, $PBr(X)$ is eliminated early.

For brevity, if $C \notin \mathcal{FHUOI}$ or C is *not closed* (or *not maximal*), then C is referred to as *a non-closed* FHUO ($NonC\_FHUO$) (or *non-maximal* FHUO, $NonM\_FHUO$) itemset. Similarly, if $C \notin \mathcal{FHUOI}$ or C is not a generator, then it is called a non-generator FHUO ($NonG\_FHUO$) itemset. If $Br(C)$ cannot include any CFHUOIs (or GFHUOIs), it is also referred to as a $NonC\_FHUO$ (or $NonG\_FHUO$) branch. In addition, if it cannot contain both CFHUOIs and GFHUOIs, it is also called a $NonC\_FHUO$ and $NonG\_FHUO$ branch, or briefly, $NonCG\_FHUO$ branch.

2. Extended pruning strategies of non-closed or non-generator branches

Even after removing infrequent itemsets and LUOIs, a QTDB can still generate numerous FHUOIs. Thus, extracting CFHUOIs and GFHUOIs is challenging, as it requires checking inclusion relationships to filter out non-closed and non-generator FHUOIs. A key challenge is how to efficiently prune $NonC\_FHUO$ and $NonG\_FHUO$ branches

from the search space. Since the occ function is monotonic within each equivalence class, $NonC\_FHUO$ branches can be pruned using strategies from [7]. However, the early elimination of $NonG\_FHUO$ branches requires an anti-monotonic function in each equivalence class, which occ lacks. To address this issue, the authors of [8] introduced the weak lower bound (WLB) $wlbocc$ on occ. For any itemset A, a function $wlb(A)$ is a WLB on $occ(A)$ if $wlb(A) \leq occ(B)$ for any proper FE B of A. The $wlbocc$ WLB is defined as follows.

Definition 13 (*WLB $wlbocc$ on occ* [8]). For any strictly frequent itemset X (i.e. $l \overset{\text{def}}{=} |h(X)| \geq min\_sup$) and transaction $T_i \in h(X)$, let us define $\mathbb{Q}(X,T_i) \overset{\text{def}}{=} u(X,T_i) + \min\left\{u\left(a_j,T_i\right) \mid a_j \in \text{rem}(X,T_i)\right\}$, and $\mathbb{Q}_{rel}(X,T_i) \overset{\text{def}}{=} \mathbb{Q}(X,T_i)/tu(T_i)$. Let $\mathcal{F} = \{\mathbb{Q}_{rel}(X,T_i) \mid T_i \in h(X)\}$ represent the set of relative weights $\mathbb{Q}_{rel}(X,T_i)$ across all transactions in $h(X)$. Assume that the series $\mathcal{F}$ is sorted in ascending order to create the new one $\mathcal{F}' = \{\tau_i, i = 1..l\}$. Then, the $wlbocc$ WLB is defined as follows:

$$wlbocc(X) = \frac{1}{min\_sup} \sum_{1 \leq i \leq min\_sup} \tau_i \qquad (12)$$

Proposition 2 (*Property of $wlbocc$* [8]). $wlbocc$ is a WLB on occ, i.e. $wlbocc(X) \leq occ(S)$ for any strictly frequent itemset X and its frequent proper FE $S = X \oplus Y, Y \neq \emptyset$.

Example 7. For $min\_sup = 2$ and $min\_uo = 0.1$, consider itemset $X = a$, and a proper FE S of X, $X \subset S = X \oplus e = ae$. We have $h(X) = \{T_1, T_4, T_5\}$, $\mathbb{Q}(X,T_1) = 16 + \min\{16\} = 32$, $\mathbb{Q}(X,T_4) = 16 + \min\{10;2;12;52\} = 18$, and $\mathbb{Q}(X,T_5) = 48 + \min\{18;55\} = 66$. Then, $\mathbb{Q}_{rel}(X,T_1) = \mathbb{Q}(X,T_1)/tu(T_1) = \frac{32}{32} = 1$, $\mathbb{Q}_{rel}(X,T_4) = \mathbb{Q}(X,T_4)/tu(T_4) = \frac{18}{92} = 0.196$, $\mathbb{Q}_{rel}(X,T_5) = \mathbb{Q}(X,T_5)/tu(T_5) = \frac{66}{121} = 0.545$ and $\mathcal{F} = \{1; 0.196; 0.545\}$. Thus, $\mathcal{F}' = \{\tau_1, \tau_2, \tau_3\} = \{0.196; 0.545; 1\}$ and $wlbocc(X) = \frac{\tau_1+\tau_2}{min\_sup} = \frac{0.196+0.545}{2} = 0.371 < occ(S) = 0.642$.

Consider an itemset S being evaluated during the mining process, such as $C = bc$, which is formed by joining $X = b$ and $Y = c$. The key challenge is determining whether the $PBr(C)$ branch contains no CFHUOIs and/or GFHUOIs, allowing it to be pruned. This decision relies on elements in the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets, as well as information from previous levels of the prefix tree, including nodes representing X, Y, and their parents. To address this, the paper leverages the following propositions. Notably, by the time $C = bc$ is evaluated, the exploration of $Br(a)$ has already been completed. This prior exploration helps identify and store numerous CFHUOI and GFHUOI candidates in their respective sets, derived from $Br(a)$. Building on theoretical results from [7] and [8], this paper extends the pruning strategies for $NonC\_FHUO$ or $NonG\_FHUO$ branches, as detailed below.

Proposition 3 (*Early pruning of $NonC\_FHUO$ or $NonG\_FHUO$ branches using backwards*). Consider the current itemset $C \in \mathcal{FHUOI}$ and two sets $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ that include the current candidates of CFHUOIs and GFHUOIs, respectively.

(i) (*EPNonCloBr - The halting of the search for CFHUOIs*) The entire branch $Br(C)$ cannot contain any CFHUOIs, i.e. it is a $NonC\_FHUO$ branch, if there exists an itemset $D \in \mathcal{CFHUOI}$ such that $D \supset C, lastItem(C) = lastItem(D)$ and $supp(D) = supp(C)$. Thus, it is unnecessary to find CFHUOIs in this branch.

(ii) (*EPNonGenBr - The halting of the search for GFHUOIs*) The branch $PBr(C)$ cannot include any GFHUOIs, i.e. it is a NonG_FHUO branch, if there exists an itemset $R \in \mathcal{GFHUOI}$ satisfying the following conditions: $R \subset C, supp(R) = supp(C)$ and $wlbocc(R) \geq min\_uo$. Therefore, we can omit the search for larger GFHUOIs generated from C.

The primary limitation of the EPNonCloBr and EPNonGenBr strategies is the need to verify inclusion relationships between itemsets, which can be computationally expensive. To address this, the paper introduces alternative pruning strategies, derived from the theoretical results in [7] and [8], as outlined in the following proposition.

Proposition 4 (*Local pruning of $NonC\_FHUO$ or $NonG\_FHUO$ branches using nodes at two successive levels of the prefix tree*). Consider two nodes $X = P \oplus x$ and $Y = P \oplus y$, which share the same non-empty parent P in the prefix tree, where $y \succ x$, and a forward extension S of X with y, $S = X \oplus y$.

(i) (*LPNonCloBr*) If $supp(S) = supp(Y)$, then $Br(Y)$ is a NonC_FHUO branch and thus we ignore discovering CFHUOIs in this branch.

(ii) (*LPNonGenBr*) If the following conditions are met: ($supp(S) = supp(Y)$ and $wlbocc(Y) \geq \min\_uo$) or ($supp(S) = supp(X)$ and $wlbocc(X) \geq min\_uo$), then $PBr(S)$ is a NonG_FHUO branch, and thus we stop the search for larger GFHUOIs generated from S. Additionally, the non-generator FHUO node corresponding to S is discarded if any of the following conditions hold: $occ(Y) \geq min\_uo, occ(X) \geq min\_uo$, or $S \notin \mathcal{FHUOI}$.

Pruning the $NonC\_FHUO$ $Br(C)$ or $NonG\_FHUO$ $PBr(C)$ branches by Proposition 4 relies solely on the support of itemsets Y (or X) and S without verifying the obvious inclusion $Y \subset S$ (or $X \subset S$, respectively).

One of the main objectives of this paper is to develop an efficient algorithm that simultaneously discovers both $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets. The key difference between the pruning strategies in Propositions 3 and 4 and those in [7] and [8] lies in how the pruning conditions are applied. Specifically, when the conditions in Propositions 3.(i) and 4.(i) (or Propositions 3.(ii) and 4.(ii)) are met, the search halts only for CFHUOIs (or GFHUOIs, respectively). In other words, if the conditions in Propositions 3.(i) and 4.(i) hold, the algorithm continues searching solely for GFHUOIs in $Br(C)$. Similarly, if the conditions in Propositions 3.(ii) and 4.(ii) are satisfied, the algorithm proceeds to mine solely CFHUOIs in $Br(C)$.

3. Novel pruning strategies

Note that the techniques in Propositions 3 and 4 are used to prune only $NonC\_FHUO$ or $NonG\_FHUO$ branches. In the following, the paper introduces novel pruning strategies to prune branches early that do not contain both CFHUOIs and GFHUOIs.

Corollary 1 (*EPNonCGBr - Early pruning of $NonCG\_FHUO$ branches using backwards*). Consider the current itemset $C \in \mathcal{FHUOI}$ and two sets $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ that include the current candidates of CFHUOIs and GFHUOIs, respectively. If there exist itemsets $D \in \mathcal{CFHUOI}$ and $R \in \mathcal{GFHUOI}$ such that the following conditions are met $R \subset C \subset D$, $lastItem(C) = lastItem(D)$, $supp(D) = supp(C) = supp(R)$, and $wlbocc(R) \geq min\_uo$, then the $PBr(C)$ branch cannot include any CFHUOIs and GFHUOIs, i.e. it is a $NonCG\_FHUO$ branch, and thus it can be pruned early.

Proof: Corollary 1 follows directly from Proposition 3. ■

Corollary 2 (*LPNonCGBr - Local pruning of $NonCG\_FHUO$ branches using nodes at three successive levels of the prefix tree*). For any two nodes having the same parent P in the prefix tree, $X = P \oplus x$ and $Y = P \oplus y$, where $y \succ x$, and an itemset $D = prefix(P) \oplus y$, consider a forward extension itemset S of X with the last item y of the node $Y, S = X \oplus y = P \oplus x \oplus y$. Then, if ($supp(S) = supp(P)$ and $wlbocc(P) \geq min\_uo$) or ($supp(S) = supp(D)$ and $wlbocc(D) \geq min\_uo$, then the $NonCG\_FHUO$ $PBr(Y)$ proper branch can be pruned early, since it cannot contain any CFHUOIs and GFHUOIs. In addition, if $occ(P) \geq min\_uo$, $occ(D) \geq min\_uo$ or $Y \notin \mathcal{FHUOI}$, then the Y node is also eliminated safely.

Proof: Since $X = P \oplus x \subset S = P \oplus x \oplus y = P \oplus xy$ and $P \subset Y = P \oplus y \subset S = P \oplus xy$, and $D = parent(P) \oplus y \subset Y = P \oplus y \subset S$, the itemsets S, Y and D share the same last item y. ■

- If $supp(S) = supp(P)$, then by Lemma 1 in [8], $supp(S) = supp(Y) = supp(P)$. By Proposition 4.($i$), $PBr(Y)$ cannot contain any CFHUOIs. In addition, if $wlbocc(P) \geq min\_uo$, then by Proposition 4.(ii), $PBr(Y)$ cannot also contain any GFHUOIs. Therefore, $PBr(Y)$ is a $NonCG\_FHUO$ branch, and thus it can be early pruned.
- Similarly, since $D = parent(P) \oplus y$, we have $D \subset Y \subset S$, the itemsets S and D share the same last item y. If $supp(S) = supp(D)$, then by Lemma 1 in [8], we have $supp(S) = supp(Y) = supp(D)$. If $wlbocc(D) \geq min\_uo$, the whole $NonCG\_FHUO$ $PBr(Y)$ proper branch is also pruned safely by Proposition 4.

The remaining assertions are proven similarly.

Example 8. Figure 1 illustrates applying the pruning techniques outlined in Proposition 4 and Corollary 2 to prune $NonC\_FHUO$ and/or $NonG\_FHUO$ branches in the prefix tree. The figure presents a portion of the tree generated using the FE operator with parameters $min\_uo = 0.18$ and $min\_sup = 1$. To improve clarity and readability, each node is labeled with an itemset rather than just its last item. Additionally, four values are assigned to each node, representing support ($supp$) and utility occupancy (occ) as superscripts, and the weak upper bound ($wubocc$) and weak lower bound ($wlbocc$) as subscripts. This notation provides a clearer and more intuitive representation of the example. For instance, the itemset $bc^{2;0.11}_{1;0.26}$ signifies that $supp(bc) = 2$, $occ(bc) = 0.11$, $wubocc(bc) = 1$ and $wlbocc(bc) = 0.26$. Consider the itemsets $X = bc^{2;0.11}_{1;0.26}$ and $Y = bd^{2;0.25}_{0.92;0.54}$, which share the same prefix $P = b^{2;0.06}_{1;0.09}$, along with the itemset $D = d^{2;0.19}_{0.9;0.53}$. When X is extended by adding d, forming a new itemset $S = X \oplus d = bcd^{2;0.3}_{1;0.63}$, it is observed that $supp(S) = supp(Y) = supp(D) = 2$ and $wlbocc(D) = 0.53 \geq min\_uo$. As a result, applying the $LPNonCGBr$ strategy from Corollary 2 enables early pruning of the $NonCG\_FHUO$ branch $PBr(Y)$ (highlighted with a red border) since it is guaranteed to contain no any CFHUOIs and GFHUOIs. Furthermore, since $occ(D) = 0.19 > min\_uo$, and despite Y being an FHUOI, the Y node, which is neither a CFHUOI nor a GFHUOI, is also pruned, along with the $NonCG\_FHUO$ branch $Br(Y)$. Similarly, when X is extended with e, forming the itemset $S_1 = X \oplus e = bce^{2;0.54}_{0.76;0;76}$, it is observed that $supp\,(S_1) = supp(be) = 2$ and $wlbocc(be) = 0.67 > min\_uo$. Consequently, according to Proposition 4, this enables early pruning of both the $NonC\_FHUO$ $Br(be)$ and $NonG\_FHUO$ $PBr\,(S_1)$ (highlighted with a blue border). Additionally, although S_1 belongs to the $\mathcal{FHUOI}$ set, since $occ(be) = 0.49 > min\_uo$, the $NonG\_FHUO$ S_1 node is also eliminated.

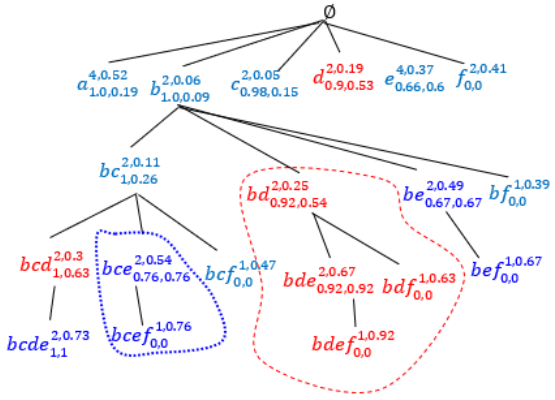

Figure 1. Illustrating strategies in Proposition 4 and Corollary 2.

4. Forming the result sets

a) The construction of the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets

After eliminating all unpromising patterns, those that are infrequent, have low utility occupancy, or are non-closed and non-generator FHUOIs, using Propositions 1-2 and Corollaries $1-2$, the remaining itemsets become strong candidates for CFHUOIs and GFHUOIs.

Let the search space for mining the $\mathcal{CFHUOI}$ set be represented as a prefix tree, where a depth-first search is performed following the predefined item order $\prec$. Initially, $\mathcal{CFHUOI}$ is first initialized as an empty set. For a CFHUOI candidate C being under consideration, it will be added to $\mathcal{CFHUOI}$, if the following condition (1) is satisfied:

$$[C \in \mathcal{FHUOI}, C \text{ has no } 1\text{- forward } and(\nexists D \in \mathcal{CFHUOI} : D \supset C \text{ and } supp(D) = supp(C))] \quad (13)$$

Note that if D is a $1-$ forward or $1-$ backward extension of C, it will either be processed after C or has already been processed before C, respectively, during prefix tree construction. For example, given $C = ce$, its $1-$forward extension is cef, while cde and bce are its $1-$backward extensions. This follows from their equal support values: $supp(cef) = supp(cde) = supp(bce) = supp(C)$. Based on (1), the following proposition [7] is used to construct the $\mathcal{CFHUOI}$ set efficiently.

Proposition 5 (*The construction of the $\mathcal{CFHUOI}$ set* [7])

$$\mathcal{CFHUOI} = \{C \in \mathcal{A} \mid C \text{ is a } CFHUOI\}$$

This proposition states that by adding itemsets C to $\mathcal{CFHUOI}$ based on the conditions in (1), the $\mathcal{CFHUOI}$ set is constructed correctly and completely. Moreover, the aforementioned process for the form of $\mathcal{CFHUOI}$ ensures that no non-closed FHUOIs are generated during the process.

For the construction of the $\mathcal{CFHUOI}$ set, which is defined as $\mathcal{GFHUOI} \stackrel{\text{def}}{=} \{A \in \mathcal{FHUOI} \mid \nexists B \in \mathcal{FHUOI} : B \subset A \wedge supp(B) = supp(A)\}$, it requires checking two conditions: (1) A belongs to the $\mathcal{FHUOI}$ set, and (2) no proper subset B of A in $\mathcal{FHUOI}$ shares the same support as A. During the mining process, each remaining itemset S is evaluated. If S is an $FHUOI$ and no itemset in $\mathcal{CFHUOI}$ is its proper subset with the same support, S is added to proper. Then, for each candidate R in $\mathcal{GFHUOI}$, if $S \subset R$ and $supp(R) = supp(S), R$ is removed since it cannot be a GFHUOI. This iterative process ensures that the $\mathcal{GFHUOI}$ set is continuously updated throughout mining. Upon completion, the remaining itemsets in $\mathcal{GFHUOI}$ constitute the final and complete set of GFHUOIs.

b) The construction of the $\mathcal{MFHUOI}$ set

As mentioned in the Introduction section, the paper aims to develop two effective algorithms to simultaneously discover both CFHUOIs and GFHUOIs at the same process, and to exploit MFHUOIs separately due to their benefits. For CFHUOIs and GFHUOIs, we use the theoretical results in Sections 4.2-4.3 to eliminate unpromising candidates, and the method in Section 4.4.a to construct the result sets $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$. For MFHUOIs, new methods are required to discover them effectively. Note that we have the following assertion $\mathcal{MFHUOI} \subseteq \mathcal{CFHUOI} \subseteq \mathcal{FHUOI}$, and thus in [7] the authors introduced the method for determining the $\mathcal{MFHUOI}$ set from $\mathcal{CFHUOI}$ as follows: $\mathcal{MFHUOI} = \{A \in \mathcal{CFHUOI} \mid \nexists B \in \mathcal{CFHUOI} : B \supset A\}$. However, since the cardinality of the $\mathcal{CFHUOI}$ set is often much larger than that of $\mathcal{MFHUOI}$, checking the inclusive relationship between a larger number of itemsets may take much more time. In addition, as the current study needs to develop a new algorithm that discovers only $\mathcal{MFHUOI}$ without $\mathcal{CFHUOI}$, constructing the $\mathcal{CFHUOI}$ set is unnecessary. Based on Proposition 3.(i), Proposition 4.(i) and the relationship $\mathcal{MFHUOI} \subseteq \mathcal{CFHUOI}$, the following corollary is derived.

Corollary 3 (*NonMaxBr Pruning strategies*).

(i) (*EPNonMaxBr - Early pruning of NonM_FHUO branches by checking backwards*) Consider the current itemset $C \in \mathcal{FHUOI}$ and the $\mathcal{MFHUOI}$ set of MFHUOI candidates. If there exists an itemset $D \in \mathcal{MFHUOI}$ such that $D \supset C$, $lastItem(C) = lastItem(D)$ and $supp(D) = supp(C)$, the entire branch $Br(C)$

cannot contain any MFHUOIs, and thus it can be pruned early.

(ii) (*LPNonMaxBr - Local pruning of NonM_FHUO branches*) Consider two nodes $X = P \oplus x$ and $Y = P \oplus y$, which share the same non-empty parent P in the prefix tree, where $y \succ x$, and a forward extension S of X with y, $S = X \oplus y$. If $supp(S) = supp(Y)$, then the $NonM\_FHUO$ $Br(Y)$ branch can be eliminated early.

Based on the definition of the $\mathcal{MFHUOI}$ set, it can be constructed by verifying two conditions: $X \in \mathcal{FHUOI}$ and $(\nexists Y \in \mathcal{FHUOI} : Y \supset X)$. For each remaining FHUOI S that is not pruned by Corollary 3 in the mining process, if no itemset in $\mathcal{MFHUOI}$ is its superset, it will be added to the $\mathcal{MFHUOI}$ set. Next, for each candidate itemset R in $\mathcal{MFHUOI}$, if the condition $R \subset S$ holds, R is removed from $\mathcal{MFHUOI}$. Hence, the $\mathcal{MFHUOI}$ is continuously updated during the mining process. When the algorithm finishes, the complete and correct $\mathcal{MFHUOI}$ set of all MFHUOIs is constructed.

Example 9. Consider two itemsets $X = bcd^{2;0.3}_{1;0.63}$ and $Y = bce^{2;0.54}_{0.76;0.76}$ in Figure 1. When X is extended with the last item e of Y to create a new itemset $S = X \oplus y = bcde^{2;0.73}_{1;1}$, it is observed that the pruning condition $supp(S) = supp(Y) = 2$ is satisfied. Thus, the whole $Br(Y)$ branch cannot contain any MFHUOIs, i.e. it is a NonM_FHUO branch, and it can be eliminated early to reduce the search space.

5. Proposed algorithms

Based on the proposed theoretical results, this section develops two novel algorithms, which are CGFHUOI-Miner to simultaneously discover both CFHUOIs and GFHUOIs in the same process, and MaxFHUOI-Miner to exploit MFHUOIs. The proposed algorithms are implemented using the MUO-list data structure, which is a modified version of UO-list in [19] to additionally store necessary information needed for discovering concise representations of FHUOIs. **Figure 2** illustrates the Pre-processing procedure, which operates on a QTDB $\mathcal{D}'$ as input and requires two threshold parameters min_uo and min_sup. Initially, the procedure scans the $\mathcal{D}'$ to compute the support of each item and the total utility (tu) value of each transaction. Based on this computation, it identifies the set E of frequent items, where each item's support is greater than or equal to min_sup (line 1). Infrequent items are then removed from $\mathcal{D}'$ to reduce QTDB, thus facilitating the subsequent item-extension process (line 2). Following this, the procedure sorts the remaining items in ascending order based on their supports to enhance the efficiency of further operations.

Procedure 1: Pre-processing.
Input: QTDB $\mathcal{D}'$, min_uo and min_sup.
Output: + Two sets $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ of all CFHUOIs and GFHUOIs if using the CGFHUOI-Miner algorithm, or
+ The $\mathcal{MFHUOI}$ set of all MFHUOIs if using the MaxFHUOI-Miner algorithm.

1. **Scan** $\mathcal{D}'$ to calculate the tu value of each transaction, and identify E, the list of frequent 1-itemsets
2. **Remove** all *infrequent* items from $\mathcal{D}'$;
3. **Scan** $\mathcal{D}'$ again to construct the structure MUO-list for all *frequent* items in E;
4. $\mathcal{CFHUOI} := \emptyset$; $\mathcal{GFHUOI} := \emptyset$; $\mathcal{MFHUOI} := \emptyset$;
5. **for** each $x \in E$ **do**
6. a. **CGFHUOI-Miner** (x, E, min_uo, min_sup, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$);
6. b. **MaxFHUOI-Miner** (x, E, min_uo, min_sup, $\mathcal{MFHUOI}$);
7. **return** ($\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$), or $\mathcal{MFHUOI}$;

Figure 2. The Pre-processing procedure.

The algorithm scans $\mathcal{D}'$ once to construct the MUO-list structure for all items in E (line 3). For each frequent item x in E, it calls the CGFHUOI-Miner algorithm (line 6.a) to simultaneously discover both CFHUOIs and GFHUOIs at the same process, or invokes the MaxFHUOI-Miner algorithm (line 6.b) to exploit only MFHUOIs. Upon completion, the procedure outputs the discovered sets, $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$, or $\mathcal{MFHUOI}$ (line 7).

The CGFHUOI-Miner algorithm. Figure 3 shows the CGFHUOI-Miner algorithm that begins by evaluating whether any pruning conditions (lines 1 and 2) are satisfied for the itemset X. If the condition in line 1 is met, the corresponding branch $Br(X)$ is pruned, and the algorithm terminates for X. If the condition in line 2 holds, the algorithm also stops after completing the procedures Update-GenFHUOI and Update-CloFHUOI if X is identified as an FHUOI (line 3). Next, strategies in Corollaries 1-2 are used to early prune $NonCG\_FHUO$ branches that cannot includes both CFHUOIs and GFHUOIs (lines 9-13). If these branches are not eliminated, the algorithm will use strategies in Propositions 3 and 4. At lines 14-19, Propositions 3.(ii) and 4.(ii) are applied to early eliminate $NonG\_FHUO$ proper branch $PBr(X)$, while Propositions 3.(i) and 4.(i) are adopted in lines 20-23 to prune $NonC\_FHUO$ branch $Br(X)$. Note that if $X.Prune\_NonG\_PBr$ is true (line 14), the algorithm only ignores discovering GFHUOIs in $PBr(X)$, but mining

Algorithm 1: CGFHUOI-Miner (X, E, min_uo, min_sup, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$)

Input: The itemset X, set E, min_uo, min_sup, and two sets $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$.

Output: $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ have been updated.

1. **if** ($supp(X) < min\_sup$) **then return;** // *Pruning //strategy based on support*
2. **if** ($wubocc(X) < min\_uo$) **then {** // *Strategy EPLUOI*
3. **if** ($occ(X) \geq min\_uo$) **then {**
4. Update-GenFHUOI (X, min_uo, $\mathcal{GFHUOI}$);
5. Update-CloFHUOI (X, min_uo, $\mathcal{CFHUOI}$);
6. **}**
7. **return**;
8. **}**
9. **if** ($X.Prune\_NonCG\_Br$ **or** ($X.Prune\_NonC\_Br$ **and** $X.Prune\_NonG\_Br$) **or** ***hasBackwardExt_CG***(X, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$)) **then** **{** // *Corollary 2 & Corollary 1*
10. **if** ($X.Prune\_NonG\_Node = false$) **then**
11. Update-GenFHUOI (X, min_uo, $\mathcal{GFHUOI}$);
12. **return;**
13. **}**
14. **if** ($X.Prune\_NonG\_PBr$) **then** { *//Propositions 3 and 4.(ii)*
15. **If** ($X.Prune\_NonG\_Node = false$) **then**
16. Update-GenFHUOI (X, min_uo, $\mathcal{GFHUOI}$);
17. **Search-CloFHUOI** (X, E, min_uo, min_sup, $\mathcal{CFHUOI}$);
18. **return**;
19. **}**
20. **If** ($X.Prune\_NonC\_Br$) **then** { *//Propositions 3 and 4.(i)*
21. **Search-GenFHUOI** (X, E, min_uo, min_sup, $\mathcal{GFHUOI}$);
22. **return**;
23. **}**
24. $P = prefix(X)$; $ListEx = \emptyset$;
25. **for each** $y \in E$ such that $y \succ lastItem(X)$ **do** {
26. **if** ($supp(Xy) \geq min\_sup$) **then** {
27. **Add** $S = X \oplus y$ to $ListEx$ and build S'MUO-list;
28. **EarlyPruning-NonCGI** (X, Py, S); // *Proposition 4 and Corollary 2*
29. **}**
30. **}**
31. **if** ($occ(X) \geq min\_uo$) **then** {
32. Update-GenFHUOI (X, min_uo, $\mathcal{GFHUOI}$);
33. Update-CloFHUOI (X, min_uo, $\mathcal{CFHUOI}$);
34. **}**
35. **for each** $S \in ListEx$ **do**
36. **CGFHUOI-Miner**(S, E, min_uo, min_sup, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$);
37. **return** $\mathcal{CFHUOI}$ **and** $\mathcal{GFHUOI}$;

Figure 3. The CGFHUOI-Miner algorithm.

CFHUOIs in this branch is still required by invoking the Search-CloFHUOI proceduce, which is similar to the FindMaxCloFHUOI procedure presented in [7]. Similarly, if $X.Prune\_NonC\_Br = true$ (line 20), mining CFHUOIs in the $Br(X)$ branch is omitted, but the Search-GenFHUOI procedure [8] is called to exploit GFHUOIs in this branch. Next, for each item $y \in E$ such that $y \succ lastItem(X)$, CGFHUOI-Miner constructs the MUO-list structure for the extension $S = Xy$ and calls the EarlyPruning-NonCGI procedure to mark for early pruning branches based on conditions shown in Proposition 4 and Corollary 2 (lines 25-30). After that, if the condition $occ(X) \geq min\_uo$ in line 31 holds, the procedures Update-GenFHUOI and Update-CloFHUOI are called to update the sets $\mathcal{GFHUOI}$ and $\mathcal{CFHUOI}$, respectively (lines 32 and 33). Finally, for each itemset S in ListEx, the CGFHUOI-Miner algorithm recursively calls itself (lines 35-36) to explore and identify larger CFHUOIs and GFHUOIs.

The MaxFHUOI-Miner algorithm. The pseudo-code of the MaxFHUOI-Miner algorithm is shown in Figure 4. Like the CGFHUOI-Miner algorithm, MaxFHUOI-Miner takes as input an itemset X, the E set of candidate items for extending X, two minimum thresholds min_uo and min_sup, and the $\mathcal{MFHUOI}$ set of promising patterns for MFHUOIs. Its output is the complete and correct $\mathcal{MFHUOI}$ set of all MFHUOIs. First, the algorithm begins with checking the pruning conditions at line 1 , and it will stop the mining process on the subtree rooted at the X node if one of the conditions is satisfied according to Corollary 3. (i). Then, if the $wubocc$ value of X is less than min_uo, the Update-MaxFHUOI procedure is called (if the condition $occ(X) \geq min\_uo$ holds) before discovering MFHUOIs in the $Br(X)$ branch is terminated.

Next, for each item y belonging to the E set such that $y \succ lastItem(X)$, if itemset $S = Xy$ is frequent, the algorithm constructs the data structure MUO-list of S. Then, it marks the $Br(Py)$ branch for early pruning if the $supp(S) = supp(Py)$ condition holds. Finally, for each itemset S belonging to the ListEx set, the MaxFHUOI-Miner algorithm recursively calls itself to discover larger MFHUOIs in the $Br(X)$ branch.

Remark (*Optimization strategy: Reducing subset checks and repeated candidate verifications in the sets* $\mathcal{CFHUOI}$ *and* $\mathcal{GFHUOI}$).

To enhance the efficiency of updating the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets, we propose an optimization strategy that minimizes redundant subset checks and repeated candidate verifications. Instead of employing support values as hash keys, an approach commonly adopted in prior studies, we utilize the TWU values of itemsets to index $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ via hash tables. Here,

Algorithm 2: MaxFHUOI-Miner (X, E, min_uo, min_sup, $\mathcal{MFHUOI}$)
Input: The itemset X, the set E, thresholds min_uo and min_sup, and $\mathcal{MFHUOI}$.
Output: The set $\mathcal{MFHUOI}$ has been updated.

1. **if** ($supp(X) < ms$ **or** $X.isPruned$ **or** *// Corollary 3. (i)*
   hasBackwardExt_Max (X, $\mathcal{MFHUOI}$)) **then**
2. **return**;
3. **if** ($wubocc(X) < min\_uo$) **then** {*// Strategy EPLUOI*
4. **if** ($occ(X) \geq min\_uo$) **then**
   Update-MaxFHUOI(X, $\mathcal{MFHUOI}$);
5. **return**;
6. }
7. $P = prefix(X)$; $ListEx = \emptyset$;
8. **for each** $y \in E$ such that $y \succ lastItem(X)$ **do** {
9. **if** ($supp(Xy) \geq min\_sup$) **then** {
10. **Add** $S = Xy$ to $ListEx$ and construct the structure MUO-list of S;
11. **if** ($supp(S) = supp(Py)$) **then**
12. $Py.isPruned = true$; *//Corollary 3. (ii)*
13. }
14. }
15. **if** ($occ(X) \geq min\_uo$) **then**
16. **Update-MaxFHUOI**(X, $\mathcal{MFHUOI}$);
17. **for each** $S \in ListEx$ **do**
18. **MaxFHUOI-Miner** (S, $newE$, min_uo, ms, $\mathcal{MFHUOI}$);
19. **return** $\mathcal{MFHUOI}$;

Figure 4. The MaxFHUOI-Miner algorithm.

Procedure 2: EarlyPruning-NonCGI (X, Y, S)
Input: Three itemsets X, Y, and S.
Output: Two itemsets Y and S have been updated.

1. $P = prefix(X)$; $D = prefix(P) \oplus lastItem(Y)$;
2. **if** (($supp(S) = supp(P)$ **and** $wlbocc(P) \geq min\_uo$) **or** ($supp(S) = supp(D)$ **and** $wlbocc(D) \geq min\_uo$)) **then** {
3. $Y.Prune\_NonCG\_Br = true$; *//Corollary 2*
4. **if** ($occ(P) \geq min\_uo$ **or** $occ(D) \geq min\_uo$ **or** $occ(Y) < min\_uo$) **then**
5. $Y.Prune\_NonG\_Node = true$; *//Corollary 2*
6. }
7. **else** {
8. **if** ($supp(S) = supp(Y)$)
9. $Y.Prune\_NonC\_Br = true$; *//Proposition 4. (i)*
10. }
11. **if** (($supp(S) = supp(Y)$ **and** $wlbocc(Y) \geq min\_uo$) **or** ($supp(S) = supp(X)$ **and** $wlbocc(X) \geq min\_uo$)) **then {**
12. $S.Prune\_NonG\_PBr = true$; *//Proposition 4. (ii)*
13. **if** ($occ(Y) \geq min\_uo$ **or** $occ(X) \geq min\_uo$ **or** $occ(S) < min\_uo$) **then**
14. $S.Prune\_NonG\_Node = true$; *//Proposition 4. (ii)*
15. }

Figure 5. The EarlyPruning-NonCGI procedure.

the TWU of an itemset X is defined as $TWU(X) = \sum_{T_i \in \rho(X)} tu(T_i)$, where $\rho(X)$ denotes the set of transactions containing X, and $tu(T_i)$ is the total utility of transaction T_i. This choice is grounded in the following insight: if an itemset X fails to satisfy the conditions for inclusion in $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$, namely, if there exists an itemset $\exists Y \in \mathcal{FHUOI}$ such that $Y \subset X$ (or $Y \supset X$) and $supp(Y) = supp(X)$, then it must hold that $\rho(Y) = \rho(X)$ and consequently $TWU(Y) = TWU(X)$. As a result, X and Y will reside in the same bucket of the TWU-indexed hash table.

Compared to support-based indexing, TWU-based hashing significantly reduces hash collisions. For example, in a scenario where $min\_uo = 0.18$ and $min\_sup = 1$, the number of discovered FHUOIs is 46. Consider itemset $X = af$, which appears in only one transaction T_5, resulting in $supp(X) = 1$ and $TWU(X) = 121$. Then, only two FHUOIs, af and aef, share the same TWU value, whereas as many as 29 FHUOIs have a support value of 1. This illustrates that TWU provides a finer granularity for distinguishing itemsets, thus reducing unnecessary comparisons.

Example 10 (*An illustration of the main steps of the CGFHUOI-Miner algorithm*).

An illustrative example demonstrating key steps of the CGFHUOI-Miner algorithm is presented in the following figures, with parameters set to $min\_uo = 0.18$ and $min\_sup = 1$, where items are arranged in lexicographic order. Initially, Procedure 1 is executed to preprocess the database $\mathcal{D}'$ by scanning each transaction to compute its transaction utility (tu) and identify the set E of valid items. As all items satisfy the minimum support threshold, no items are removed from $\mathcal{D}'$ (line 2 of Procedure 1). Subsequently, for each item $x \in E = \{a, b, c, d, e, f\}$, the CGFHUOI-Miner algorithm is invoked to discover CFHUOIs and GFHUOIs from the corresponding branch $Br(x)$, which includes x and its forward extensions. In the execution trace for branch $Br(A)$ illustrated in Figure 8, none of the pruning conditions at lines $1, 2, 9, 14$, and 20 of Algorithm 1 are satisfied. Consequently, item x is extended with items b, c, d, e, f to generate the $2-$itemsets ab, ac, ad, ae, af (lines 25-29), which are not pruned by the condition at line 26. Furthermore, when the EarlyPruningNonCGI procedure is invoked at line 28

Procedure 3: Update-CloFHUOI (C, $\mathcal{CFHUOI}$)
Input: The FHUOI C and the set $\mathcal{CFHUOI}$.
Output: $\mathcal{CFHUOI}$ has been updated.
1. **for** each $D \in \mathcal{CFHUOI}$ **do**
2. **if** ($C \subset D$ and $supp(C) = supp(D)$) **then**
3. **return**;
4. **for** each $R \in \mathcal{CFHUOI}$ **do**
5. **if** ($R \subset C$ and $supp(R) = supp(C)$) **then**
6. *Remove* R from $\mathcal{CFHUOI}$;
7. $\mathcal{CFHUOI} \coloneqq \mathcal{CFHUOI} \cup \{C\}$;

Procedure 4: Update-GenFHUOI (C, $\mathcal{GFHUOI}$)
Input: The FHUOI C and the set $\mathcal{GFHUOI}$.
Output: $\mathcal{GFHUOI}$ has been updated.
1. **for** each $D \in \mathcal{GFHUOI}$ **do**
2. **if** ($D \subset C$ and $supp(C) = supp(D)$) **then**
3. **return;**
4. **for** each $R \in \mathcal{GFHUOI}$ **do**
5. **if** ($C \subset R$ and $supp(R) = supp(C)$) **then**
6. Remove R from $\mathcal{GFHUOI}$;
7. $\mathcal{GFHUOI} \coloneqq \mathcal{GFHUOI} \cup \{C\}$;

Figure 6. The Update-CloFHUOI and Update-GenFHUOI procedures.

Procedure 5: *hasBackwardExt_CG*(C, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$)
Input: The itemset C and two sets $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$.
Output: the *True* value is returned, if C has a superset in $\mathcal{CFHUOI}$ and a subset in $\mathcal{GFHUOI}$ with the same support; otherwise, it returns *False*.
1. **for** each $D \in \mathcal{CFHUOI}$ **and** $R \in \mathcal{GFHUOI}$ **do** *//Corollary 1*
2. **if** ($R \subset C \subset D$ **and** $lastItem(C) = lastItem(D)$ **and** $supp(C) = supp(D) = supp(R)$ **and** $wlbocc(R) \geq min\_uo$) **then**
3. **return** true; *//Corollary 1*
4. **return** false;

Figure 7. The hasBackwardExt_CG procedure.

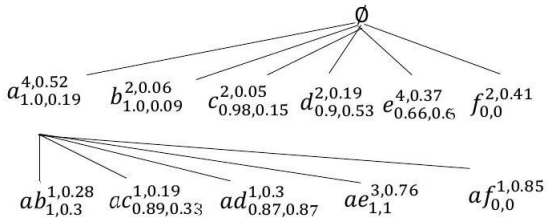

Figure 8. Execution of CGFHUOI-Miner on the $Br(a)$ branch.

for each generated 2−itemset, no branches are pruned in accordance with Proposition 4 and Corollary 2. Since the condition $occ(X) \geq min\_uo$ is satisfied (line 31), item X is added to both the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets through the Update-CloFHUOI and Update-GenFHUOI procedures (lines 32-33), resulting in $\mathcal{CFHUOI} = \{a\}$ and $\mathcal{GFHUOI} = \{a\}$.

Next, the execution of CGFHUOI-Miner for the branch $Br(X)$, $X = ab$, is examined in Figure 9. As the pruning conditions at lines 1, 2, 9, 14, and 20 in Algorithm 1 are not satisfied, the $Br(X)$ branch cannot be pruned and is retained for further exploration. When X is extended with the last item $y = c$ from itemset $Y = ac$, resulting in the new itemset $S = X \oplus y = abc$ (line 27), Procedure 2 (EarlyPruning-NonCGI) in Figure 5 is invoked to determine whether any branches can be eliminated early based on the proposed pruning strategies. Within Procedure 2, it is found that $supp(S) = supp(Y) = 1$ (line 8) and $wlbocc(Y) = 0.33 \geq min\_uo$ (line 11). Hence, the discovery of CFHUOIs in the $Br(Y)$ branch is skipped (line 9) according to Proposition 4. (i), and the branch is marked for early pruning in subsequent calls to CGFHUOI-Miner. Furthermore, since $occ(Y) = 0.19 > min\_uo$ (line 13), the branch $Br(S)$ is also marked for pruning as a $NonG\_FHUO$ branch, following Proposition 4. (ii) (lines 12 and 14). Similarly, when X is extended with items d and e, producing the itemsets abd and abe, the corresponding branches $Br(abd)$ and $Br(ad)$ are marked as $NonG\_FHUO$ and $NonC\_FHUO$, respectively, and are pruned in accordance with Propositions 4. (i) and 4. (ii). It is also noted that the itemset abf is not generated, as it does not occur in any transaction (i.e., $supp(abf) = 0$). Finally, since ab satisfies the required conditions, it is added to both $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ by lines 32−33 of Algorithm 1, resulting in the updated sets $\mathcal{CFHUOI} = \{a, ab\}$ and $\mathcal{GFHOUI} = \{a, ab\}$.

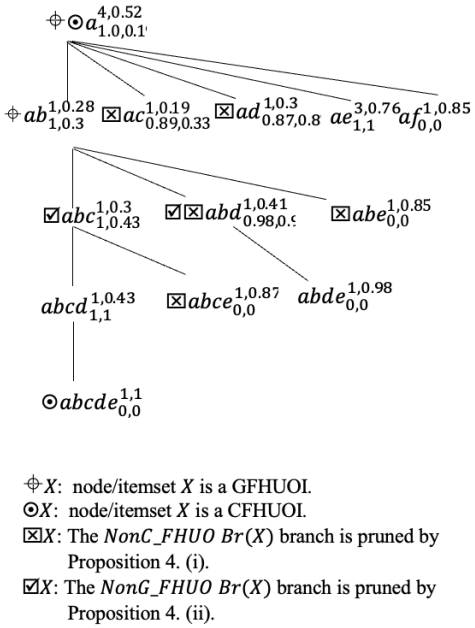

Figure 9. Execution of CGFHUOI-Miner on the $Br(a)$ branch.

We now examine the execution of CGFHUOI-Miner for the itemset $X = abc$, as illustrated in Figure 9. Since the branch $Br(X)$ has been previously marked as containing no GFHUOI, only the Search-CloFHUOI procedure is invoked to identify CFHUOIs within this branch (line 17). When X is extended with items d and e, forming the itemsets $abcd$ and $abce$, Procedure 2 (Figure 5) is called. As a result, the branches $Br(abcd)$ and $Br(abce)$ are marked as NonC_FHUO and pruned early, as they cannot contain any CFHUOIs. Subsequently, since $occ(X) \geq min\_uo$ (line 31), the procedures Update-CloFHUOI and Update-GenFHUOI are executed to update the $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$ sets, respectively (lines 32-33). During the execution of Update-CloFHUOI, the itemset ab is removed from $\mathcal{CFHUOI}$ and abc is added, because $R = ab \subset C = X = abc$ and $supp(R) = supp(C) = 1$, where $R \in \mathcal{CFHUOI}$ (lines 4-7 in Procedure 3). Meanwhile, the $\mathcal{GFHUOI}$ set remains unchanged. The updated sets are thus $\mathcal{CFHUOI} = \{a, abc\}$ and $\mathcal{GFHUOI} = \{a, ab\}$. Similarly, when CGFHUOI-Miner is recursively invoked for $X = abcd$ (see Figure 9), the itemset $abcde$ is generated, and the branch $Br(abce)$ is marked as $NonC\_FHUO$ for early pruning. Then, abc is removed from $\mathcal{CFHUOI}$ and $abcd$ is added via the UpdateCloFHUOI procedure. Following the same procedure, during the execution of CGFHUOI-Miner with $X = abcde$, the itemset $abcd$ is removed from $\mathcal{CFHUOI}$, and $abcde$ is added. At this point, the final sets become $\mathcal{CFHUOI} = \{a, abcde\}$ and

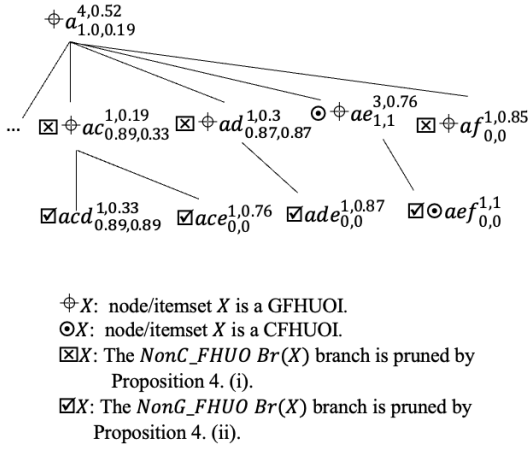

Figure 10. Execution of CGFHUOI-Miner on the $Br(ac), Br(ad), Br(ae)$ and $Br(af)$ branches.

$\mathcal{GFHUOI} = \{a, ab\}$. It is also important to note that the recursive calls to CGFHUOI-Miner for $X = abce$ and $X = abd$ result in no further exploration, as their respective branches have been previously marked as $NonCG\_FHUO$. According to Corollaries 1 and 2, these branches satisfy the condition in line 9 of Algorithm 1 and are therefore pruned early.

Subsequently, during the execution of CGFHUOI-Miner for $X = abe$, followed by $X = ac$ and $X = ad$ (see Figure 10), the branches $Br(X)$ are early pruned (line 20 of Algorithm 1), since they were previously marked as $NonC\_FHUO$ for elimination. As a result, only GFHUOIs need to be discovered in these branches through the invocation of the SearchGenFHUOI procedure (line 21 of Algorithm 1). Given that $supp(abe) = supp(ab) = 1$, the itemset abe is not added to the $\mathcal{GFHUOI}$ set. In contrast, the itemsets ac and ad satisfy the required conditions and are thus added to $\mathcal{GFHUOI}$, resulting in $\mathcal{GFHUOI} = \{a, ab, ac, ad\}$. Furthermore, the branches $Br(acd), Br(ace)$ and $Br(ade)$ are marked for early pruning as $NonG\_FHUO$, according to the LPNonGenBr pruning strategy defined in Proposition 4. (ii). In the subsequent execution of CGFHUOI-Miner for $X = ae$, when X is extended with item f to generate the itemset aef, $Br(aef)$ and $Br(af)$ are marked as $NonG\_FHUO$ and $NonC\_FHUO$, respectively, for early elimination. Consequently, the itemsets ae and aef are added to the $\mathcal{CFHUOI}$ set, while ae and af are included in the $\mathcal{GFHUOI}$ set.

Upon completing the execution of CGFHUOI-Miner for the branch $Br(A)$, the resulting sets are $\mathcal{GFHUOI} = \{a, ab, ac, ad, ae, af\}$ and

$\mathcal{CFHUOI} = \{a, abcde, ae, aef\}$. Similar results are obtained from the execution of CGFHUOI-Miner on the remaining branches $Br(b), Br(C), Br(d), Br(e)$, and $Br(f)$.

Finally, the complete result sets are: $\mathcal{GFHUOI} = \{a, ab, ac, ad, ae, af, be, bf, ce, cf, d, df, e, f\}$ and $\mathcal{CFHUOI} = \{a, abcde, ae, aef, bcde, bcdef, e, ef\}$.

Time complexity of the proposed algorithms. This paragraph begins by analyzing the worst-case time complexity of the MaxFHUOI-Miner algorithm, which is primarily influenced by the number of forward extensions. Let N denote the total number of items in the set $\mathcal{A}$, and let $n = \max\{|T| : T \in \mathcal{D}\}$ be the maximum transaction length. For any itemset X with length $L = |X|$, and where $a_i = lastItem(X)$ and $L \leq i \leq N$, the recursive algorithm MaxFHUOI-Miner$(X, E, min_uo, min_sup, \mathcal{MFHUOI})$, or simply MaxFHUOI-Miner (X), can generate up to $FE(i) \stackrel{\text{def}}{=} N - i$ forward extensions, as indicated in Line 15 of Algorithm 2 in Figure 4. At step L, where $1 \leq L \leq n$, let $T(L, i)$ represent the computational cost of executing MaxFHUOI-Miner (X) for a specific itemset X, and let $\mathcal{T}(L)$ denote the total cost for all itemsets X with length $|X| = L$. Let us define $FHUOI(L)$ as the set of all frequent high utility occupancy itemsets of length L, with the convention that $|FHUOI(0)| = 1$. Additionally, the maximum number of forward extensions possible for all itemsets of length L is given by $ME(L) \stackrel{\text{def}}{=} \sum_{i=L..N} FE(i) = (N - L)(N - L + 1)/2 = O\left(N^2\right)$. Then, $T(L, i) = FE(i) + \sum_{k=i+1..N} T(L + 1, k)$ for $L = 1, \ldots, n-1, \mathcal{T}(L) = |FHUOI(L-1)| \times \sum_{i=L..N} T(L, i)$, and $T(n, i) \stackrel{\text{def}}{=} 0, \mathcal{T}(n) \stackrel{\text{def}}{=} 0$. Moreover, $T(n-1, i) = \text{FE}\ (i) = N - i = O(N - i), \mathcal{T}(n-1) = |FHUOI(n-2)| \times ME(n-1) = |\ FHUOI(n-2)| \times (N-n+1)(N-n+2)/2 = |FHUOI(n-2)\ | \times O\left((N - n + 2)^2\right)$. Similarly, we can derive that $T(n-2, i) = (N-i)(N-i+1)/2 = O\left((N-i)^2\right)$, and the total complexity at level $n-2$ is given by $\mathcal{T}(n-2) = |FHUOI(n-3)| \times [(N-n+2)(N-n+3)(N-n+4)/6] = |FHUOI(n-3)| \times O\left((N - n + 2)^3\right)$. In general, using induction, we obtain $T(L, i) = O\left((N-i)^{n-L}\right)$ and $\mathcal{T}(L) = |FHUOI(L-1)| \times O((N-n+2)^{n-L+1})$. Specifically, the worst-case time complexity of MaxFHUOI-Miner is $\mathcal{T}(1) = O\left((N - n + 2)^n\right)$. By the same reasoning, the worst-case complexity of CGFHUOI-Miner is also $O\left((N - n + 2)^n\right)$. Notably, when $n = N$, we have $\mathcal{T}(1) = 2^N - 1 = O\left(2^N\right)$, which corresponds to the total number of non-empty subsets of the set $\mathcal{A}$ containing N items. However, when early pruning conditions are satisfied (e.g., as in Lines 1, 3, and 11 of Algorithm 2), such as when $wubocc(X) < min\_uo$ in Line 3, all forward extensions of X can be safely eliminated. In this case, the number of pruned itemsets can reach up to $T(L, i) = O\left((N - i)^{n-L}\right)$. This implies that the earlier (i.e., for smaller values of L) and more frequently the pruning conditions hold, the greater the number of unpromising candidates that are eliminated. In essence, these pruning strategies significantly reduce the search space, which explains the strong performance of both MaxFHUOI-Miner and CGFHUOI-Miner compared to existing state-of-the-art algorithms.

V. EXPERIMENTAL EVALUATION

This section presents a detailed performance analysis of the proposed algorithms in comparison to both baseline and previously established methods for mining concise representations of FHUOIs. The experiments were conducted on a system running Windows 10, equipped with an Intel(R) Core(TM) i5-5200U 2.2 GHz CPU and 12 GB of RAM. A key highlight of this study is that CGFHUOI-Miner and MaxFHUOI-Miner are the first algorithms designed to discover both CFHUOIs and GFHUOIs at the same process, as well as to mine only MFHUOIs, respectively. To ensure a comprehensive evaluation, MaxFHUOI-Miner was compared against the baseline algorithm BL-MaxFHUOI-Miner, which first extracts the complete set of FHUOIs from the dataset using SHO [6], and then identifies all MFHUOIs in a post-processing step. Furthermore, the performance of CGFHUOI-Miner was assessed against CloFHUOIM [7] and Gen-FHUOIM [8][9], denoted as CloFHUOIM+Gen-FHUOIM, focusing on runtime efficiency and memory consumption. Notably, CloFHUOIM was developed specifically for extracting CFHUOIs, while Gen-FHUOIM was designed for mining GFHUOIs. To ensure a fair and meaningful comparison, the runtime of CGFHUOI-Miner was benchmarked against the combined runtime of CloFHUOIM and Gen-FHUOIM for each tested min_sup and min_uo value pair. Likewise, the maximum memory usage was determined by evaluating the peak memory consumption of CloFHUOIM and Gen-FHUOIM.

All algorithms were implemented in Java 8 SE to ensure a consistent and reliable platform for performance evaluation. The experiments were conducted using synthetic datasets generated by the IBM Quest data generator (as referenced in [30]), which provides extensive control over key parameters, including average transaction length(T), average size of maximal frequent itemsets (I), number of distinct items (N), and dataset size (D) (measured in thousands of transactions). To comprehensively assess algorithm performance, evaluations were carried out on both real-world QDBs, Connect, Chess, Mushroom, and BMS, and synthetic QTDBs T20I4N600D- and C20D10K. The real-world QTDBs and C20D10K were obtained from [30], while T20I4N600D- was generated using the IBM

Quest data generator. The tested QTDBs exhibit a range of density levels: Connect and Chess are highly dense, Mushroom and C20D10K are moderately dense, while BMS and T20I4N600D- are sparse. Table 5 summarizes their key characteristics, including the number of transactions, number of distinct items, average transaction length, density, and data type.

Table V
CHARACTERISTICS OF THE DATASETS

Dataset	#Trans. (D)	#Items (N)	Average trans. length (T)	Type of data
Chess	3,196	76	37	Real-life
Connect	67,557	129	43	Real-life
Mushroom	8,124	119	23	Real-life
c20d10K	10,000	192	20	Synthetic
T20I4N600D-	100K-160K	600	20	Synthetic
BMS	59,602	497	2.51	Real-life

1. Performance evaluation of the proposed algorithms

This subsection evaluates and compares the performance of the proposed MaxFHUOI-Miner and CGFHUOI-Miner algorithms against the baseline and previous approaches, focusing on runtime and memory usage across four datasets: Mushroom, Chess, BMS, and C20D10K.

a) Performance of the MaxFHUOI-Miner algorithm

The results in Figure 11 show that the runtime of the MaxFHUOI-Miner algorithm decreases as *min_sup* and *min_uo* increase. This trend is expected, as higher values of *min_sup* and *min_uo* significantly reduce the size of the discovered result sets (as shown in Figure 13).

Notably, MaxFHUOI-Miner outperforms the baseline algorithm by a factor of 10 to 21 times (in runtime) on three tested datasets, Mushroom, BMS, and C20D10K on average, particularly when *min_sup* and *min_uo* are set to lower values. On the Chess dataset, it runs approximately three times faster than BL-MaxFHUOI-Miner across all tested values of *min_sup* and *min_uo*. This significant speedup is due to the efficient pruning strategies employed by MaxFHUOI-Miner, which enable the early elimination of many unpromising candidates that cannot be MFHUOIs. By effectively reducing the search space, the algorithm not only accelerates the mining process but also minimizes memory consumption.

In contrast, BL-MaxFHUOI-Miner takes a less efficient approach: it first generates a large set of all FHUOIs and then examines each itemset individually to determine whether it qualifies as an MFHUOI. If an itemset does not meet the criteria, only a single node/itemset is removed at a time. This sequential pruning process is computationally intensive, leading to significantly higher time and memory consumption.

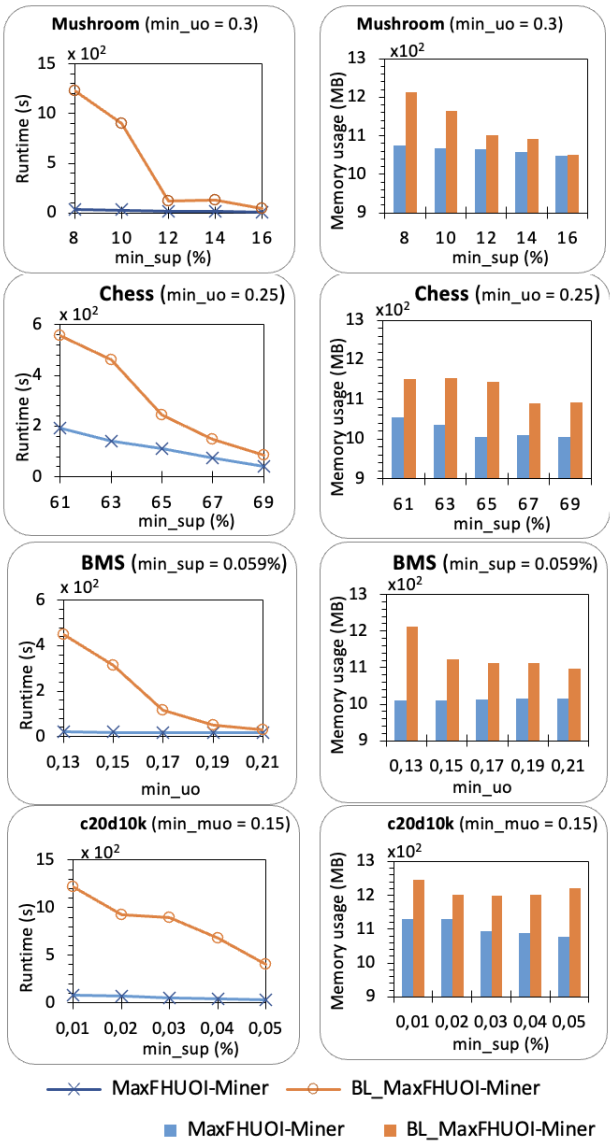

Figure 11. The performance of the MaxFHUOI-Miner algorithm.

b) Performance of the CGFHUOI-Miner algorithm

The results in Figure 12 clearly demonstrate the efficiency and effectiveness of the CGFHUOI-Miner algorithm in simultaneously discovering both CFHUOIs and GFHUOIs. By leveraging the novel EPNonCGBr and LP-NonCGBr pruning strategies, introduced in Corollary 1 and Corollary 2, the algorithm efficiently filters out non-closed and non-generator FHUOIs at an early stage, leading to significant reduction in both runtime and memory consumption across all datasets.

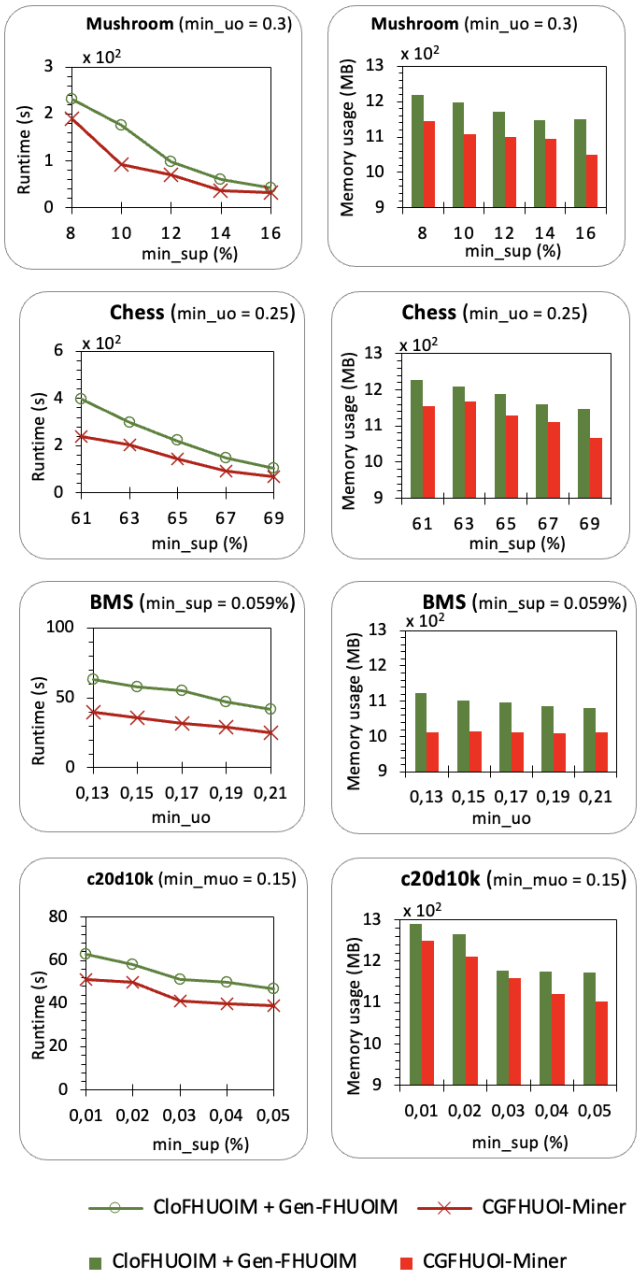

Figure 12. The performance of the CGFHUOI-Miner algorithm.

Unlike the previous CloFHUOIM + GenFHUOIM approach, which separately extracts CFHUOIs (the CloFHUOIM algorithm) and GFHUOIs (the Gen-FHUOIM algorithm), CGFHUOI-Miner performs both tasks in a single process, resulting in substantial computational savings. The notable improvements are observed in datasets such as Mushroom and BMS, where redundant patterns significantly increase computational costs.

Additionally, the proposed algorithm also stays efficient across various *min_sup* and *min_uo* thresholds, showing its strength in different mining conditions.

2. Comparison of result sets from the proposed and previous algorithms

As noted in the Introduction, the concise representation sets, $\mathcal{CFHUOI}$, $\mathcal{GFHUOI}$ and $\mathcal{MFHUOI}$, are significantly smaller than the complete $\mathcal{FHUOI}$ set, which contains all FHUOIs. These compact representations enable faster mining, reduced memory usage, and improved interpretability. To demonstrate their effectiveness on both real and synthetic datasets, Figure 13 compares the number of CFHUOIs, GFHUOIs, and MFHUOIs with the total number of all FHUOIs, mined using SHO [6], a state-of-the-art algorithm for extracting all FHUOIs. To verify correctness, the CFHUOIs and GFHUOIs generated by CGFHUOI-Miner are compared with the outputs of CloFHUOIM [7] and Gen-FHUOIM [8][9], respectively. Likewise, MFHUOIs produced by MaxFHUOI-Miner are validated against those obtained from BL_MaxFHUOI-Miner, which filters MFHUOIs from the output of the SHO algorithm. The identical results confirm the correctness and completeness of the proposed algorithms.

As shown in Figure 13, the numbers of CFHUOIs, GFHUOIs, and MFHUOIs are consistently much smaller than the number of all FHUOIs across the four tested datasets, Mushroom, Chess, BMS, and c20d10k, under various values of *min_sup* and *min_uo*. For instance, on c20d10k, GFHUOIs, CFHUOIs, and MFHUOIs account for only 4.63%, 1.65%, and 0.2%, respectively, of all FHUOIs. Notably, $\mathcal{MFHUOI}$ is the most compact, representing just 4.2% and 11.8% of the sizes of $\mathcal{CFHUOI}$ and $\mathcal{GFHUOI}$. These results emphasize the efficiency and practicality of mining concise representations over extracting the complete $\mathcal{FHUOI}$ set.

3. Scalability

This section analyzes the scalability of the compared algorithms by modifying the parameters of the synthetic dataset, as outlined in Table 5, along with the database sizes. In Figure 14 and Figure 15, a '-' symbol is added to the dataset name to indicate changes in the preceding parameter. The synthetic datasets for these different parameter settings were generated using the IBM Quest Synthetic Data Generator. The figures illustrate a comparative evaluation of the scalability of MaxFHUOI-Miner and CGFHUOI-Miner in terms of runtime and memory consumption on two large QTDBs from Table 5, Connect and T20I4N600D-. To preserve the density characteristics of each test database, the values of *min_sup* and *min_uo* thresholds remain unchanged

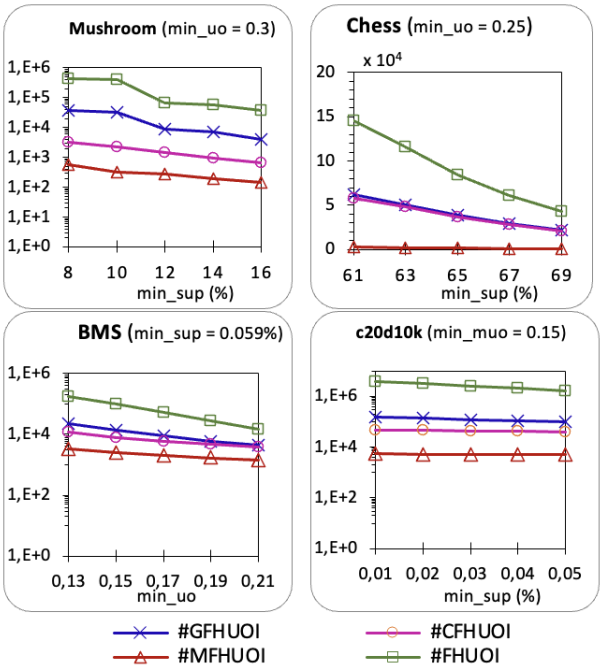

Figure 13. The number of FHUOIs and concise representations.

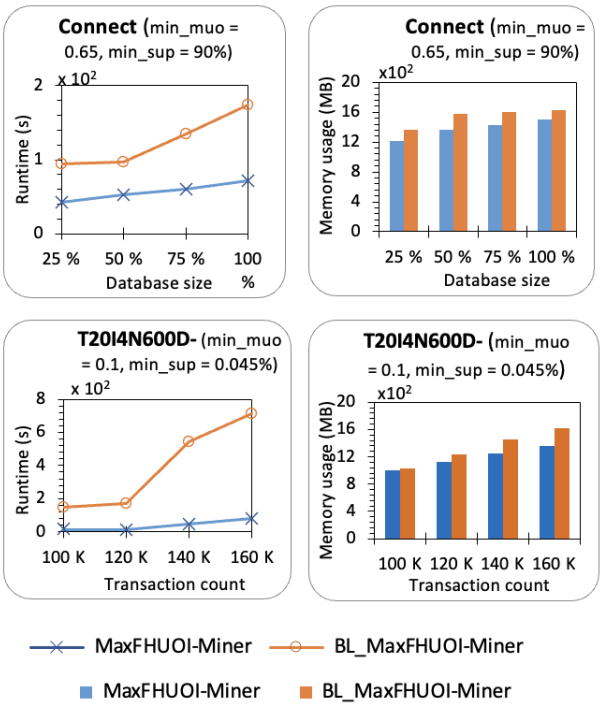

Figure 14. The scalability of the MaxFHUOI-Miner algorithm.

while the database size varies. Specifically, the size of the Connect dataset is adjusted to 25%, 50%, 75%, and 100%, while the transaction count of T20I4N600D- is modified from 100K to 120K, 140K, and 160K. In terms of runtime, the proposed algorithms demonstrate superior scalability as dataset size and transaction count increase, whereas the runtime of baseline and previous algorithms rises at a significantly faster rate. A comprehensive evaluation shows that the proposed algorithms are faster than existing methods and use significantly less memory. The rationale behind this observation aligns with the explanation provided in Subsection 5.1.

Overall, the results strongly validate the practicality and scalability of the proposed algorithms CGFHUOI-Miner and MaxFHUOI-Miner in mining concise representations of FHUOIs. The integration of advanced pruning strategies not only enhances computational efficiency but also optimizes memory utilization, making CGFHUOI-Miner and MaxFHUOI-Miner highly efficient solutions for discovering these representations. These advantages position the algorithms as state-of-the-art approaches for discovering informative and compact patterns of FHUOIs, making it promising methods for real-world high-utility occupancy itemset mining applications.

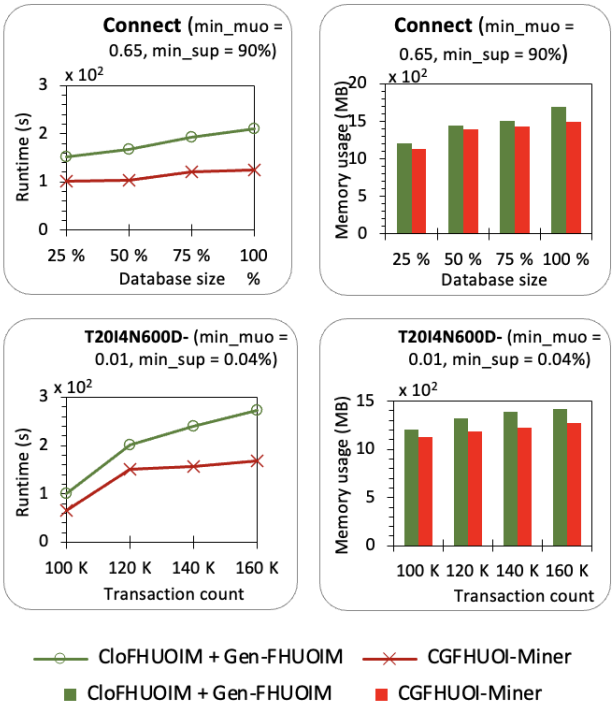

Figure 15. The scalability of the CGFHUOI-Miner algorithm.

VI. CONCLUSIONS AND FUTURE WORK

This paper introduces two novel algorithms: MaxFHUOI-Miner, for discovering only MFHUOIs, and CGFHUOI-Miner, for simultaneously mining both CFHUOIs and GFHUOIs in a single process. To efficiently prune branches containing non-closed, non-generator, or non-maximal FHUOIs in the prefix tree, these algorithms incorporate novel and extended pruning strategies. The LPNonMaxBr strategy operates at two consecutive levels in the prefix search tree, enabling the early elimination of non-maximal FHUOIs, while the LPNonCGBr strategy leverages information from three successive levels to quickly discard branches that cannot contain both CFHUOIs and GFHUOIs. Notably, these strategies eliminate the need to verify subset relationships among itemsets, significantly reducing computational costs. Additionally, the EPNonMaxBr and EPNonCGBr strategies are introduced to facilitate the early elimination of redundant patterns through backward checking. Experimental evaluations demonstrate that the proposed algorithms outperform baseline and existing methods in terms of runtime, memory efficiency, and scalability.

Looking ahead, we plan to develop algorithms for extracting concise representations of frequent high average-utility and utility occupancy sequences in quantitative sequence databases.

ACKNOWLEDGMENT

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.05-2021.52.

REFERENCES

[1] Z. H. Deng, "Mining high occupancy itemsets," *Future Generation Computer Systems*, vol. 102, pp. 222–229, 2020.

[2] K. Zhang, Y. Zhang, and Z. Wang, "Frequent pattern mining based on occupation and correlation," in *Proc. IEEE 3rd Int. Conf. Electronic Information and Communication Technology (ICEICT)*, pp. 161–166, 2020.

[3] H. Kim *et al.*, "Mining high occupancy patterns to analyze incremental data in intelligent systems," *ISA Transactions*, vol. 131, pp. 460–475, 2022.

[4] L. T. T. Nguyen, T. Mai, G. H. Pham, U. Yun, and B. Vo, "An efficient method for mining high occupancy itemsets based on equivalence class and early pruning," *Knowledge-Based Systems*, vol. 267, Art. no. 110441, 2023.

[5] B. Shen, Z. Wen, Y. Zhao, D. Zhou, and W. Zheng, "OCEAN: Fast discovery of high utility occupancy itemsets," in *Lecture Notes in Computer Science*, pp. 354–365, 2016.

[6] J. He, X. Han, J. Wang, and K. Zhang, "Efficient high-utility occupancy itemset mining algorithm on massive data," *Expert Systems with Applications*, vol. 210, Art. no. 118329, 2022.

[7] H. Duong, H. Pham, T. Truong, and P. Fournier-Viger, "Efficient algorithms to mine concise representations of frequent high utility occupancy patterns," *Applied Intelligence*, 2024.

[8] D. Hai and T. Tin, "An efficient algorithm for mining frequent high utility occupancy generators," in *Proc. 9th Int. Conf. Cloud Computing and Internet of Things (CCIOT)*, pp. 101–109, 2025.

[9] D. Hai, H. Tien, and T. Tin, "Mining concise representations of frequent high utility occupancy itemsets using generator patterns," *Dalat University Journal of Science*, to be published, 2025.

[10] J. Li, H. Li, L. Wong, J. Pei, and G. Dong, "Minimum description length principle: Generators are preferable to closed patterns," in *Proc. 21st AAAI Conf. Artificial Intelligence*, pp. 409–414, 2006.

[11] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proc. SIAM Int. Conf. Data Mining*, pp. 482–486, 2004.

[12] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proc. Int. Workshop Utility-Based Data Mining*, pp. 90–99, 2005.

[13] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. ACM Int. Conf. Information and Knowledge Management*, pp. 55–64, 2012.

[14] V. S. Tseng, C. Wu, B. Shie, and P. S. Yu, "UP-Growth: An efficient algorithm for high utility itemset mining," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 253–262, 2010.

[15] S. Zida *et al.*, "EFIM: A highly efficient algorithm for high-utility itemset mining," in *Proc. Mexican Int. Conf. Artificial Intelligence (MICAI)*, pp. 530–546, 2015.

[16] S. Krishnamoorthy, "HMiner: Efficiently mining high utility itemsets," *Expert Systems with Applications*, vol. 90, pp. 168–183, 2017.

[17] J.-F. Qu *et al.*, "Mining high utility itemsets using prefix trees and utility vectors," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 10224–10236, 2023.

[18] V. S. Tseng, C. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining the concise and lossless representation of high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, pp. 726–739, 2015.

[19] W. Gan *et al.*, "HUOPM: High-utility occupancy pattern mining," *IEEE Transactions on Cybernetics*, vol. 50, pp. 1195–1208, 2020.

[20] C. M. Chen *et al.*, "Discovering high utility-occupancy patterns from uncertain data," *Information Sciences*, vol. 546, pp. 1208–1229, 2021.

[21] S. Vemulapalli and S. Mogalla, "High utility-occupancy sequential pattern mining algorithm based on utility-occupancy framework," *Int. J. Engineering Trends and Technology*, vol. 69, pp. 228–235, 2021.

[22] C.-W. Wu, P. Fournier-Viger, J. Gu, and V. S. Tseng, "Mining compact high utility itemsets without candidate generation," in *High-Utility Pattern Mining*, Studies in Big Data, pp. 283–307, 2019.

[23] H. Duong *et al.*, "Efficient algorithms for mining closed and maximal high utility itemsets," *Knowledge-Based Systems*, vol. 257, Art. no. 109921, 2022.

[24] C.-W. Wu, P. Fournier-Viger, J.-Y. Gu, and V. S. Tseng, "Mining closed high utility itemsets without candidate generation," in *Proc. Conf. Technologies and Applications of Artificial Intelligence*, pp. 187–194, 2015.

[25] P. Fournier-Viger *et al.*, "EFIM-Closed: Fast and memory-efficient discovery of closed high-utility itemsets," in *Proc. Int. Conf. Machine Learning and Data Mining in Pattern Recognition*, pp. 199–213, 2016.

[26] L. T. T. Nguyen *et al.*, "An efficient method for mining high utility closed itemsets," *Information Sciences*, vol. 495, pp. 78–99, 2019.

[27] P. Fournier-Viger, C.-W. Wu, and V. S. Tseng, "Novel concise representations of high utility itemsets using generator

patterns," in *Proc. Int. Conf. Advanced Data Mining and Applications*, pp. 30–43, 2014.
[28] C.-W. Lin, T.-P. Hong, and W.-H. Lu, "An effective tree structure for mining high utility itemsets," *Expert Systems with Applications*, vol. 38, 2011.
[29] H. Duong, T. Truong, B. Le, and P. Fournier-Viger, "CG-FHAUI: An efficient algorithm for mining succinct pattern sets of frequent high average utility itemsets," *Knowledge and Information Systems*, 2024.
[30] P. Fournier-Viger *et al.*, "SPMF: A Java open-source pattern mining library," *Journal of Machine Learning Research*, vol. 15, pp. 3569–3573, 2014.

Tien Hoang received his MSc degree in Computer Science from the University of Science, VNUHCMC, Vietnam, in 2009. He is currently a lecturer in the Department of Mathematics and Computer Science at Dalat University, Vietnam, and a research student at the University of Science, Ho Chi Minh City. His research focuses on artificial intelligence and data mining.
Email: tienhoang@dlu.edu.vn

Lan Huynh received her Master's degree in Computer Science from the University of Science, VNU-HCMC, in 2011. She is currently a lecturer in the Faculty of Information Technology at Ho Chi Minh City University of Industry and Trade, Vietnam. Her research focuses on artificial intelligence and data mining.
Email: lanhuynh@dlu.edu.vn

Hai Duong is an Associate Professor at Dalat University, Vietnam. He obtained his Ph.D. in Computer Science in 2020 from the University of Science, VNU-HCMC. During his doctoral studies, he received several awards for outstanding scientific research. Currently, he serves as both a lecturer and researcher in the Faculty of Mathematics and Computer Science at Dalat University, where he also holds the position of Vice Head. His primary research interests include artificial intelligence and data mining.
Email: haidv@dlu.edu.vn

Tin Truong is a researcher at the Department of Mathematics and Computer Science, University of Dalat, Vietnam. He received his B.S. degree in Mathematics from Dalat University in 1983 and his Ph.D. in Stochastic Optimal Control in 1990 at the Vietnam National University, VNU-Hanoi, Vietnam. His current research interests are artificial intelligence and data mining.
Email: tintruong@dlu.edu.vn