Hindawi
Complexity
Volume 2021, Article ID 7310137, 18 pages
https://doi.org/10.1155/2021/7310137

Research Article

Efficient Utility Tree-Based Algorithm to Mine High Utility Patterns Having Strong Correlation

Rashad Saeed[1,2] **Azhar Rauf**[1] **Fahmi H. Quradaa**[1,2] **and Syed Muhammad Asim**[3]

[1]*Department of Computer Science, University of Peshawar, Peshawar, Pakistan*
[2]*Department of Computer Science, Aden Community College, Aden, Yemen*
[3]*Department of Statistics, University of Peshawar, Peshawar, Pakistan*

Correspondence should be addressed to Rashad Saeed; rashadsaeedstd@uop.edu.pk

Received 18 May 2021; Accepted 12 July 2021; Published 28 July 2021

Academic Editor: Dan Selişteanu

High Utility Itemset Mining (HUIM) is one of the most investigated tasks of data mining. It has broad applications in domains such as product recommendation, market basket analysis, e-learning, text mining, bioinformatics, and web click stream analysis. Insights from such pattern analysis provide numerous benefits, including cost cutting, improved competitive advantage, and increased revenue. However, HUIM methods may discover misleading patterns as they do not evaluate the correlation of extracted patterns. As a consequence, a number of algorithms have been proposed to mine correlated HUIs. These algorithms still suffer from the issue of the computational cost in terms of both time and memory consumption. This paper presents an algorithm, named Efficient Correlated High Utility Pattern Mining (ECoHUPM), to efficiently mine the high utility patterns having strong correlation items. A new data structure based on utility tree (UTtree) named CoUTlist is proposed to store sufficient information for mining the desired patterns. Three pruning properties are introduced to reduce the search space and improve the mining performance. Experiments on sparse, very sparse, dense, and very dense datasets indicate that the proposed ECoHUPM algorithm is efficient as compared to the state-of-the-art CoHUIM and CoHUI-Miner algorithms in terms of both time and memory consumption.

1. Introduction

We live in a data age where a huge amount of data is generated from different devices every day. It is expected that 463 exabytes of data will be generated on a daily basis by 2025 [1]. Data mining has received a great deal of attention in order to transform data into useful information, due to the exponentially explosive growth of data [2]. Pattern mining is a type of unsupervised data mining approach, which aims to find useful, interesting, and meaningful patterns that can be used to support decision-making [3, 4]. Different pattern mining techniques are used to mine different types of patterns, including frequent patterns [5], high utility patterns [6], sequential patterns, trends, outliers, and graph structures [2, 6].

Frequent itemset mining (FIM) aims to extract patterns containing items that frequently appear in transactional database. [7]. This task has been tremendously studied and remains to this day a very active research area as it has several applications in domains such as market basket analysis, product recommendation, text mining, e-learning, bioinformatics, and web click stream analysis [3, 8, 9]. Even though the mining of frequent pattern is useful, it depends on the assumption that all items in the dataset are equally important (e.g., weight or profit). Nevertheless, this assumption is not true for several real-life applications [6, 10]. For instance, the pattern $\{\text{bread}, \text{milk}\}$ in a transaction database may be extremely frequent but it may not be interesting as it may produce a low profit. In different circumstances, numerous patterns like $\{\text{champagne}, \text{caviar}\}$ may yield a higher profit even if they are not frequent [11]. To overcome this limitation of FIM, an emerging research area is High Utility Itemset Mining (HUIM) which aims to find high utility or important patterns [2, 6].

HUIM takes into account the weight of items in the database and their quantities in each transaction. The goal of HUIM is to find all patterns having utility not less than minimum utility threshold. Recently, HUIM has become a very active research area as it generalizes the problem of FIM and has the same wide applications [12–15].

The algorithms of HUIM are divided into two main categories. The first category is called Two-Phase algorithms [11, 16]. These types of algorithms generate candidates in the first phase, and then, in the second phase, they calculate the utility of each candidate in order to derive HUIs. However, due to the huge number of candidates generated in the first phase, these algorithms may suffer from the problem of time and memory consumption. The second category is One-Phase algorithms [6, 10, 17]. The algorithms of this category try to overcome the above issue by utilizing different data structures to store sufficient information for mining the desired patterns without candidate's generation and utilize various pruning properties to reduce the search space.

One critical downside of High Utility Itemset Mining methods is that they generally extract patterns with a high utility, but the items that make up these patterns are weakly correlated. For marketing decisions, such patterns are either useless or misleading [18–23]. For instance, with market basket analysis application, the current algorithm of High Utility Pattern Mining may find that buying a pen and a 60-inch plasma TV is a high utility itemset, since these items generally create a high profit when purchased together. However, these items are weakly correlated and rarely sold together. Hence, it would be a mistake to use this pattern to promote TV to customers who buy pen [11, 21].

To address the above-stated issue, few numbers of algorithms have been developed to mine Correlated High Utility Itemsets, such as HUIPM [19], FDHUP [21], FCHMbond [22], FCHMall-confidence [22], CoHUIM [20], and CoHUI-Miner [24]. These algorithms differ from each other in the measures used to evaluate the interestingness of the extracted patterns, the data structures, and pruning properties that they used to reduce the search space and improve the mining performance. In [20, 24], a projected database has been utilized to reduce the database and improve the efficiency of correlated HUIs mining. The projected database is effective, but it suffers from the computational cost in terms of running time and memory consumption.

In order to address such issue in mining Correlated High Utility Itemsets, this study proposes a new algorithm named Efficient Correlated High Utility Pattern Mining (ECoHUPM). In the proposed algorithm, new efficient data structures and pruning properties are introduced to mine the desired patterns in efficient manner. The main contributions of this paper are summarized as follows:

(i) It proposes a novel algorithm, ECoHUPM, which adopts the divide-and-conquer approach and employs UTtree structure which is an extended form of FP-tree [25].

(ii) New data structure based on UTtree named CoUTlist is proposed to store sufficient information for mining the desired patterns in one phase without candidate's generation.

(iii) The proposed algorithm introduces several pruning properties to reduce the search space and improve the mining performance.

(iv) An experimental performance evaluation of the proposed algorithm is conducted on sparse, very sparse, dense, and very dense datasets. The performance of the proposed ECoHUPM algorithm is compared with CoHUIM and CoHUI-Miner algorithms for Correlated High Utility Itemset Mining. Experimental results show that the proposed ECoHUPM algorithm is better than the state-of-the-art CoHUIM and CoHUI-Miner algorithms in terms of both time and memory consumption.

The rest of this paper is organized as follows: In Section 2, we review the literature associated with HUIM and CoHUIM. Next, we introduce the mathematical preliminaries and state the problem in Section 3. In Section 4, we explain the proposed algorithm in detail. Section 5 gives details of the experimental setup and analyzes the results. Section 6 concludes the work of this paper.

2. Related Works

This section reviews the literature on HUIM and the CoHUIM.

2.1. High Utility Itemset Mining (HUIM).

2.1. High Utility Itemset Mining (HUIM). Yao and Hamilton defined the problem of HUIs mining in 2004 [26]. They developed UMining algorithm for mining the itemset having high utility. UMining is an approximate algorithm and may fail to extract all HUIs. Hence, in order to extract the complete set of HUIs, Liu et al. [16] developed a Two-Phase algorithm. In the Two-Phase algorithm, a novel upper bound pruning property named TWU (Transaction Weighted Utilization) has been proposed to reduce the search space. The Two-Phase algorithm mines the HUIs in two phases. In the first phase, it generates the candidate HUIs with their TWU not less than the minimum utility threshold. Then, in the second phase, it calculates the utility of each candidate by scanning the database again to drive the HUIs. However, the Two-Phase algorithm suffers from the issue of time and memory efficiency. The main reason is that a huge number of candidates may be generated in the first phase.

In [27], a new method based on tree structure called HUP-tree is proposed to mine HUIs. It integrates the Two-Phase procedure and FP-tree concept to construct a compressed tree structure for utilizing the TWU property. This approach mines HUIs in three steps: (1) constructs the tree, (2) generates the candidate's patterns, and then (3) identifies the HUIs from the list of candidates. The mining performance of this algorithm is affected by the number of conditional trees constructed during the whole mining process and the traversal cost of each conditional tree. Hence, this algorithm suffers from the time and memory consumption due to the generation of a huge number of conditional trees and candidate patterns as well [28].

In order to improve the efficiency of HUIs mining, several algorithms have been developed. To extract HUIs without candidates generation, Liu and Qu proposed HUI-Miner algorithm [29]. HUI-Miner utilizes utility-list structure to store sufficient information for mining the HUIs in one phase. Then Fournier-Viger et al. developed an algorithm named FHM [30], which introduced EUCS (Estimated Utility Cooccurrence Structure) and EUCP (Estimated Utility Cooccurrence Pruning) to improve the HUIs mining performance. HUP-Miner [31] extended the HUI-Miner to speed up utility list by utilizing a look-ahead strategy and pruning the search space by database partitioning. Chen and An [32] proposed PHU-Miner which is a parallel version of HUI-Miner. A novel algorithm named ULB-Miner was developed [33], in which improved utility list has been proposed, called utility-list buffer, for speeding up the utility-list join operation and reducing the memory consumption. A new projection-based algorithm, named MAHI [34], has been proposed to speed up the discovery of HUIs by utilizing a MAprun (Matrix-based pruning strategy).

For mining HUIs without the need to set the minimum utility threshold, Tseng et al. [35] developed two types of efficient algorithms named TKU (mining Top-K Utility itemsets) and TKO (mining Top-K utility itemsets in One phase) to extract top-K high utility itemsets. However, they remain expensive in terms of both runtime and memory usage. Hence, Duong et al. [12] designed a novel algorithm named kHMC to extract the top-K HUIs more effectively. The kHMC utilizes three strategies called COV, RIU, and CUD to reduce the search space and thus improves the mining performance. Recently, Gunawan et al. [36] developed an algorithm based on binary particle swarm optimization for optimizing the search for HUIs without setting the minimum utility threshold beforehand. Instead, the minimum utility threshold is determined as a postprocessing step.

Although High Utility Pattern Mining has several applications, it has some limitations. As a consequence, many extensions of High Utility Pattern Mining appeared in the literature such as Incremental Utility Mining [37, 38] which aims to extract HUPs from dynamic databases, On-Shelf High Utility Pattern Mining [39–41] in which the shelf time of items is considered, and Concise Representations of High Utility Patterns (e.g., Maximal Itemsets [42, 43] and Closed High Utility Itemsets [44–47]) that aim to extract a small list of meaningful HUPs.

2.2. Correlated High Utility Itemset Mining (CoHUIM). A number of correlation measures have been suggested in the data mining literature which are used for association analysis, such as bond, all-confidence, any-confidence [48, 49], coherence [50] and Kulczynsky [51]. As the traditional algorithms of High Utility Itemset Mining do not consider the correlation of the extracted patterns, they may lead to noninteresting or misleading patterns. In such a case, they usually discover itemsets having high utility, but these itemsets may contain weakly correlated items.

In order to extract more interesting patterns and to avoid misleading patterns resulting from the traditional methods of HUIs mining, a number of algorithms have been proposed to mine Correlated High Utility Itemset by utilizing both utility and correlation measures. Ahmed et al. [19] first proposed an algorithm named High Utility Interesting Pattern Mining (HUIPM) with strong frequency affinity for mining interesting patterns in high utility itemset, in which the relation among items is meaningful. The HUIPM algorithm introduced a new data structure named Utility Tree based on Frequency Affinity (UTFA) as an efficient data structure to store sufficient information required for mining the desired patterns. While a new pruning property named Knowledge Weighted Utilization (KWU) has been proposed in this algorithm to reduce the search space, the HUIPM algorithm recursively creates a number of conditional trees to generate candidates and then derive interesting patterns. This procedure is time-consuming. Thus, Lin et al. [21] developed a new algorithm named fast algorithm for mining discriminative high utility patterns (FDHUP) to improve HUIPM. In the FDHUP algorithm, two data structures called Element Information table (EI table) and Frequency Utility table (FU table) have been proposed to store required information for mining the DHUP efficiently. New pruning property is based on summation of affinitive utility and the remaining affinitive utility has been introduced to reduce the search space.

Fournier-Viger et al. [22] developed Fast Correlated High Utility Itemset Miner (FCHM) algorithm for integrating the concept of correlation in High Utility Itemset Mining in order to extract profitable patterns that are highly correlated. Two versions of the algorithm have been proposed, FCHMbond and FCHMall-confidence, which are based on bond and all-confidence measures that are already used for measuring frequent correlated patterns [48, 50, 52]. The FCHM algorithm is based on HUI-Miner [29], in which the utility-list structure has been utilized, while TWU and strategy based on summation of initial and remaining utility have been used as pruning properties to reduce the search space. Moreover, FCHMbond and FCHMall-confidence utilize the antimonotonicity property of the bond and all-confidence measures, respectively, for further improving the mining performance.

Gan et al. [18, 20] proposed two algorithms to extract correlated purchase behaviors by considering the correlation and utility measures. The first algorithm [20] is named Correlated High Utility Itemset Mining (CoHUIM), while the second one [18] is Correlated high Utility Pattern Miner (CoUPM). Both algorithms use the Kulczynsky (abbreviated as Kulc) measure [51] in conjunction with utility measure to evaluate the interestingness of the desired patterns. The CoUPM utilizes the utility-list structure which is introduced in [29] as a data structure to store information required to mine the desired patterns. Meanwhile, an efficient projection mechanism and a sorted downward closure property are developed in CoHUIM to reduce the database size.

Vo et al. [24] suggested an algorithm, called CoHUI-Miner, to efficiently extract Correlated High Utility Itemset. The CoHUI-Miner applies the database projection mechanism to reduce the database size. Furthermore, it proposes a new concept called the prefix utility of projected transactions to directly calculate the utility of itemset.

Table 1 shows a summary of the Correlated High Utility Itemset Mining algorithms and their features.

3. Fundamental Concepts

This section presents preliminary concepts related to the problem of Correlated High Utility Itemset Mining. We adopted the definitions presented in previous work [53].

Definition 1 (quantitative database). Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items and for each item $i_p \in I\,(1 \le p \ge m)$ profit unit (External Utility) denoted as $\mathrm{pr}(i_p)$ in each transaction each item is associated with internal utility (Quantity) denoted as $q(i_p, T_d)$. A quantitative database $D = \{T_1, T_2, \ldots, T_n\}$ contains a set of transactions.

Table 2 shows the transactional database, while Table 3 shows external utilities for the items in Table 2.

Definition 2. Utility of an item i_p in each transaction T_d is denoted by $u(i_p, T_d)$ and is defined as $u(i_p, T_d) = q(i_p, T_d) \times \mathrm{pr}(i_p)$, where $\mathrm{pr}(i_p)$ is the external utility of an item i_p and $q(i_p, T_d)$ is the quantity of an item i_p in transaction T_d.

For example, $u(b, T_1) = 6 \times 4 = 24$.

Definition 3. Utility of an itemset X in the transaction T_d is denoted by $u(X, T_d)$ and is defined as $u(X, T_d) = \sum_{i_p \in X\,\&\,X \in T_d} u(i_p, T_d)$, that is, the sum of the utilities of all items inside pattern X in transaction T_d.

For example, $u(bc, T_1) = u(b, T_1) + u(c, T_1) = (6 \times 4) + (2 \times 4) = 32$.

Definition 4. Utility of an itemset X in the quantitative database D is denoted by $u(X)$ and is defined as $u(X) = \sum_{X \in T_d\,\&\,T_d \subseteq D} u(X, T_d)$, that is, the sum of the utilities of itemset X in all transactions containing it.

For example, $u(bc) = u(bc, T_1) + u(bc, T_2) + u(bc, T_7) + u(bc, T_8) = 32 + 32 + 24 + 20 = 108$.

Definition 5. An itemset X is called high utility itemset if $u(X) \ge \min \mathrm{Util}$, where minUtil is the minimum utility threshold.

For example, for the data presented in Table 2 with $\mathrm{minUtil} = 90$, $\{bc\}$ is high utility itemset.

Definition 6. Utility of a transaction T_d is denoted by $tu(T_d)$ and is defined as the sum of the utilities of all items inside transaction T_d. $tu(T_d) = \sum_{i_p \in T_d} u(i_p, T_d)$.

For example, the utility of transaction T_8 is calculated as $tu(T_8) = u(b, T_8) + u(c, T_8) + u(d, T_8) + u(e, T_8) = 20 + 12 + 3 + 25 = 60$.

Definition 7. The Transaction Weighted Utilization (TWU) of an itemset X in database D is defined as $\mathrm{TWU}(X) = \sum_{X \in T_d\,\&\,T_d \subseteq D} tu(T_d)$.

For example, $\mathrm{TWU}(bc) = tu(T_1) + tu(T_2) + tu(T_7) + tu(T_8) = 70 + 31 + 26 + 60 = 187$.

Definition 8. An itemset X is called High Transaction-Weighted Utilization Itemset (HTWUI) if $\mathrm{TWU}(X) \ge \min \mathrm{Util}$, where minUtil is the minimum utility threshold.

For example, with $\mathrm{minUtil} = 90$, an itemset (bc) is HTWUI.

Different measures have been used to evaluate the interestingness of the HUIs, such as frequency affinity, bond, all-confidence, and Kulczynsky. Kulczynsky measure was recommended in [2] and has been used in [18, 20, 24]. Kulczynsky (abbreviated as Kulc) is a null-invariant measure; it is not influenced by the null transactions and is used to evaluate the inherent correlation of patterns [48, 51].

Definition 9 (support). The support of an itemset X in the transactional database D is denoted by $\sup(X)$ and is defined as the proportion of transactions in the database which are matched by X. $\mathrm{Sup}(X) = (\mathrm{count}(X)/n)$, where n is the total number of transactions in the database.

For example, for the data in Table 2, $\sup(a) = (8/11) = 0.727$.

Definition 10. The correlation between items inside an itemset X based on the Kulc measure is defined as the mean of the conditional probabilities of items:

$$\mathrm{Kulc}(X) = \frac{1}{k} \sum_{i_p \in X} \frac{\sup(X)}{\sup(i_p)}, \qquad (1)$$

where k is the number of items inside X.

For example, for the data in Table 2, Kulc $(ab) = 1/2((\sup(ab)/\sup(a)) + (\sup(ab)/\mathrm{sup}p(b))) = (1/2)((0.181/0.727) + (0.181/0.363)) = 0.375$.

Definition 11. Correlated High Utility Itemset). For a given quantitative database D with minimum utility threshold (minUtil) and minimum correlation threshold (minCor), the Correlated High Utility Itemset is an itemset X such that $u(X) \ge \min \mathrm{Util}\ \&\ \mathrm{Corr}(X) \ge \min \mathrm{Corr}$.

4. Proposed Algorithm

In order to address the need for more efficient algorithm for mining Correlated High Utility Itemsets, we propose a new algorithm named Efficient Correlated High Utility Pattern Mining (ECoHUPM). This section presents the proposed ECoHUPM algorithm in detail, the data structures that it utilizes to store sufficient information for mining the desired patterns, and the pruning properties that are used to reduce the search space and improve the mining performance.

4.1. Database Revising. The proposed ECoHUPM algorithm revises the input database in its first step. First, Property 1 [16] is used to remove all 1-itemsets with their TWU less than minimum utility threshold. For instance, for the data presented in Table 2, with $\mathrm{minUtil} = 90$, "g" item is removed as its $\mathrm{TWU}(g) = tu(T_3) + tu(T_9) = 27 + 4 = 31 < \min \mathrm{Util}$.

Table 1: Summary of Correlated High Utility Itemset Mining algorithms and their features.

No. of phases	Algorithm	Measures	Data structures	Pruning properties
Two phases	HUIPM [19]	Utility and FA	UTFA	KWU
	CoHUIM [20]		Projected database	TWU
One phase	FDHUP [21]	Utility and Kulczynsky	EI table with FU table	(1) TWU (2) Sum of AU and RAU
	FCHMbond [22]	Utility and bond	Utility list	(1) TWU (2) Sum of iutil and rutil (3) Antimonotonicity of bond
	CoHUIM [20]		Projected database	TWU
	CoUPM [18]	Utility and Kulczynsky	Utility list	(1) TWU (2) Sum of iutil and rutil
	CoHUI-Miner [24]		Projected database with prefix utility	

Table 2: A transactional database.

TID	Transaction
T_1	{a: 2, b: 6, c: 2, d: 2, e: 6}
T_2	{a: 3, b: 2, c: 3, d: 2}
T_3	{a: 3, c: 2, d: 2, e: 3, g: 3}
T_4	{a: 4, c: 4, d: 4, e: 12, f: 12}
T_5	{a: 3, c: 4, d: 3, f: 4}
T_6	{a: 2, c: 3, d: 2, f: 3}
T_7	{b: 2, c: 4, d: 2}
T_8	{b: 5, c: 3, d: 3, e: 5}
T_9	{d: 3, g: 1}
T_{10}	{a: 6, c: 5}
T_{11}	{a: 2, c: 2}

Table 3: External utility.

a	b	c	d	e	f	g
3	4	4	1	5	4	1

Table 4: Items' support.

c	d	a	b	e	f	g
10	9	8	4	4	3	2

Table 5: The revised database.

TID	Transaction	TWU
T_1	{c: 8, d: 2, a: 6, b: 24, e: 30}	70
T_2	{c: 12, d: 2, a: 9, b: 8}	31
T_3	{c: 8, d: 2, a: 9, e: 15}	34
T_4	{c: 16, d: 4, a: 12, e: 60, f: 40}	132
T_5	{c: 16, d: 3, a: 9, f: 16}	44
T_6	{c: 12, d: 2, a: 6, f: 12}	32
T_7	{c: 16, d: 2, b: 8}	26
T_8	{c: 12, d: 3, b: 20, e: 25}	60
T_9	{d: 3}	3
T_{10}	{c: 20, a: 18}	38
T_{11}	{c: 8, a: 6}	14

Second, in each transaction, the utility of each item is computed through the formula quantity$\times$ profit as is stated in Definition 2. Third, items in each transaction are sorted in the descending order of their support and the total utility is assigned to each transaction. Table 4 shows the items in the descending order of their support, while Table 5 shows the revised database.

4.2. *Search Space.*

The proposed ECoHUPM algorithm utilizes a set-enumeration tree as a search space, whose efficiency has been verified in pattern mining [29]. Reversed depth-first search traversal is adopted as shown in Figure 1 to facilitate the search tree. Note that the ECoHUPM uses the support descending order to revise database and then to construct the UTtree. Hence, with reversed depth-first search, the mining order for the running example is $\{f \prec e \prec b \prec a \prec d \prec c\}$.

Definition 12. Given a set-enumeration tree and itemset X represented by a node N, a set of nodes with their ancestors N are called the extensions (supersets) of X.

For the k-itemset (itemset containing k items), we denote its extensions containing $(k+i)$ items as i-extension of the itemset. By adopting reverse depth-first traversal, any extension of itemset X is a combination of X with the item(s) before X.

For instance, in the set-enumeration tree represented in Figure 1, itemset $\{debf\}$ is 2-extension of $\{bf\}$, while itemset $\{cdebf\}$ is 3-extension of $\{bf\}$.

4.3. *Utility Tree and Correlation Utility-List Structures.*

Once the database is revised, the proposed ECoHUPM algorithm constructs the utility tree (UTtree). A UTtree is a concise structure that stores sufficient information for facilitating the mining of Correlated High Utility Itemsets in a single phase. It is an extended form of FP-tree [25], where each node consists of four fields: label, interLink, parentLink, and utList. The label refers to the item's label,interLink points to the next node of the same item, parentLink points to the parent node, and utList is a dictionary that stores both a transaction's ID as keys and item's utility in each transaction as values.

A UTtree is constructed with only one scan of the revised database as is shown in Algorithm 1. First, the tree is initialized by creating the Root node. Then the transactions are processed one by one, as shown in lines 1 and 2. The information of each transaction is inserted into the tree by calling insertTree$(T_i, \text{itemsList}, N\ d)$ function as shown in

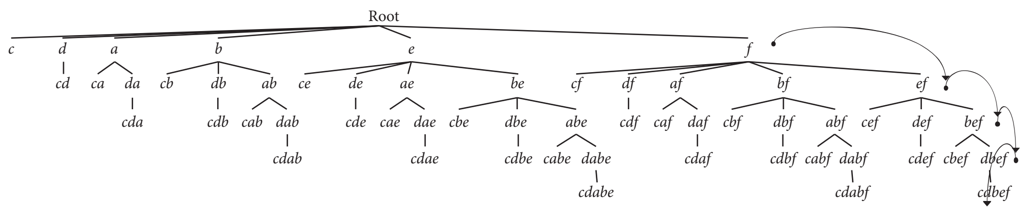


FIGURE 1: Set-enumeration tree with reversed depth-first traversal.

line 3. The function insertTree$(T_i, \text{itemsList}, N\ d)$ is executed as follows: First, the items of the current transaction T_i are sorted in itemsList, and the first item is stored in firstItem. Then, if the current node $(N\ d)$ has N child such that N.label = firstItem, update N.utList dictionary by adding T_i key with $u(\text{fisrtItem}, T_i)$ value; else create node N such that N.label = fisrtItem, N.parentLink points to $N\ d$, N.interLink points to the previous node of the same label, and $N.\text{utList} = \{T_i\colon u(\text{fisrtItem}, T_i)\}$. Thereafter, while the list of remaining items (itemsList[1:]) is not empty, call the function insertTree$(T_i, \text{remainingItems}, N)$ recursively. Finally, after inserting the information of the last transaction, the final UTtree is returned (line 5).

For the revised database presented in Table 4, Figure 2 shows the UTtree after inserting the first transaction. First, the tree is initialized by the Root node, and then node c is created with label = c, parentLink = Root, and utList = $\{T_1\colon 8\}$. Then, node d is created with *label* = d, parentLink = c, and utList = $\{T_1\colon 2\}$. Node a is created with label = a, parentLink = d, and utList = $\{T_1\colon 6\}$. Node b is created with *label* = b, parentLink = a, and utList = $\{T_1\colon 24\}$. Node e is created with label = e, parentLink = b, and utList = $\{T_1\colon 30\}$. As the current transaction is the first transaction, interLink of all nodes points to the items holding the same label in the header table. Similarly, the second and third transactions are inserted into the UTtree as shown in Figure 3. The final UTtree after inserting the last transaction is shown in Figure 4.

Besides adopting UTtree, new condensed data structure named CoUTlist is proposed to store sufficient information for mining the superset patterns without needing to scan the UTtree multiple times.

Definition 13. The Correlation Utility list (CoUTlist) of an itemset X contains a set of elements, where each element represents node called a suffix where itemset X appears. In the CoUTlist, each element has four fields:

(i) nodeNo is a unique identifier number for each node, which is used as a sequence number, for example, (1, 2, . . ., n).

(ii) nodeSupp stores the total number of transactions, where itemset X occurred in the current node N. nodeSupp = count(N.utList.keys).

(iii) nodetUtility stores the total utility of itemset X in the current node N. nodetUtility = sum(N.utlist.values).

(iv) prefixPath is a dictionary, where the keys are the labels of the parent nodes of the current node, and the values are the summation of the utilities of each parent node in the current path. prefixPath = {Parent$_1$: value$_1$, Parent$_2$: value$_2$, . . . , Parent$_n$: value$_n$}, where Parent$_i$ is the parent node in the current path and value$_i$ = sum(Parent$_i$.utList.values) for the keys in N.utList.keys.

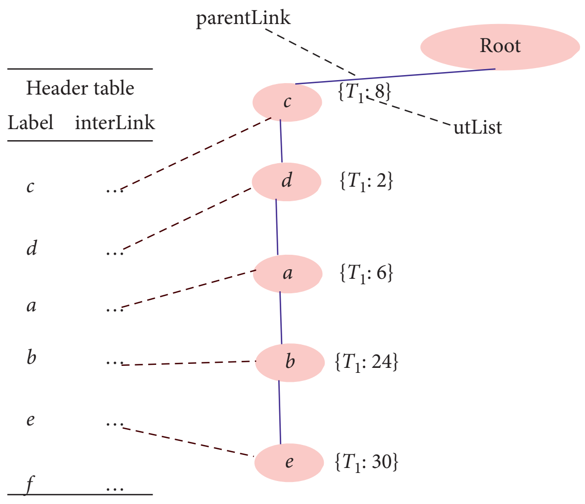


FIGURE 2: UTtree after inserting T_1.

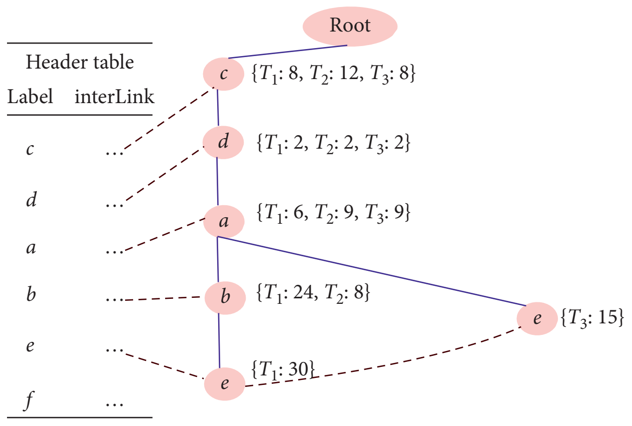


FIGURE 3: UTtree after inserting T_3.

4.3.1. The CoUTlist of 1-Itemset. Given a UTtree and item i_p, we denote the set of nodes labeled i_p as suffixes and the set of the parent nodes in the current path as prefixPath. The CoUTlist of 1-itemset (i_p) is denoted as CoUTlist(i_p) and is

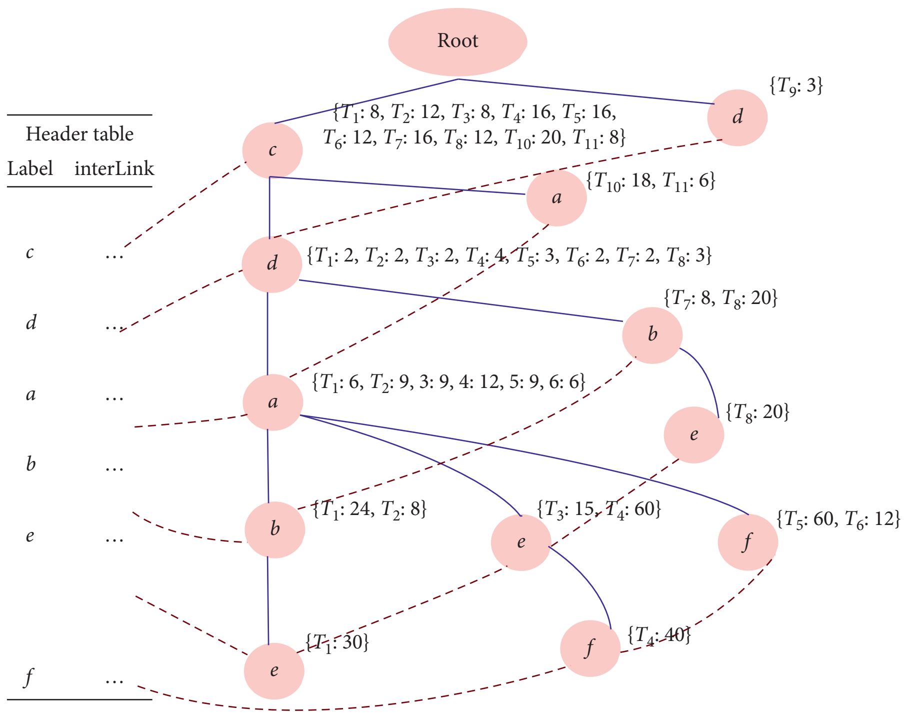

Root
Header table
Label interLink
c …
d …
a …
b …
e …
f …
{T_1: 8, T_2: 12, T_3: 8, T_4: 16, T_5: 16, T_6: 12, T_7: 16, T_8: 12, T_{10}: 20, T_{11}: 8}
{T_9: 3}
{T_{10}: 18, T_{11}: 6}
{T_1: 2, T_2: 2, T_3: 2, T_4: 4, T_5: 3, T_6: 2, T_7: 2, T_8: 3}
{T_7: 8, T_8: 20}
{T_1: 6, T_2: 9, 3: 9, 4: 12, 5: 9, 6: 6}
{T_8: 20}
{T_1: 24, T_2: 8}
{T_3: 15, T_4: 60}
{T_5: 60, T_6: 12}
{T_1: 30}
{T_4: 40}

FIGURE 4: The final UTtree after inserting T_{11}.

constructed as follows: $\forall N \in$ suffixes, if $[P_1, P_2, \ldots, P_n] \in$ prefixPath, $[P_1, P_2, \ldots, P_n]$ are the parents of N; then construct a quadruple (nodeNo, nodeSupp, nodeUtility, prefixPath), and append it to CoUTlist(i_p).

Figure 5 shows how the CoUTlist of an item $\{a\}$ denoted as CoUTlist(a) is constructed. Based on the UTtree shown in Figure 3, there are two nodes: for those whose N.label $= a$, we denote the first node as suffix1 and the second node as suffix2. According to Definition 13, the first element of the CoUTlist(a) represents suffix1, with nodeNo$\,=1$, nodeSupp = count$(N$.utList.kesy$)$ = 6, nodeUtility = sum $(N$.utList.values$)$ = $(6+9+9+12+9+6) = 51$, and pre $ffixPath$ $ffixPath = \{d\colon (u(d,T_1) + (u(d,T_2) + (u(d,T_3) + (u(d,T_4) + (u(d,T_5) + (u(d,T_6), c\colon (u(c,T_1) + (u(c,T_2) + (u(c, T_3) + (u(c,T_4) + (u(c,T_5) + (u(c,T_6)\} = \{d\colon 15, c\colon 72\}$. The second element of the CoUTlist(a) represents suffix2, with $no\ de\ No = 2$, nodeSupp = count$(N$.utList.kesy$)$ = 2, nodeUtility = sum$(N$.utList. $values$ $values)$ = $(18+6) = 24$, and $path = \{c\colon (u(c,T_{10}) + (u(c,T_{11})\} = \{c\colon 28\}$. Subsequently, the elements of suffix1 and suffix2 formalized CoUTlist(a) as shown in Figure 6.

In the same manner, the CoUTlist of the remaining 1-itemsets are constructed as shown in Figure 7.

4.3.2. The CoUTlist of k-Itemset. Let itemset $X = \{I_k, I_{k-1}, \ldots, I_2, I_1\}$ be an extension of itemset $Y = \{I_{k-1}, \ldots, I_2, I_1\}$. We denote the element in CoUTlist(Y) as Y_{node} and the element in CoUTlist(X) as X_{node}.

Definition 14. Item x_i is after itemset X if x_i is after all items in X. Here the keys {represent items} in prefixPath of Y_{node} are sorted according to the mining order which is the ascending order on their supports and are denoted as $X \prec x_i$.

For example, we have the first node of the CoUTlist of f itemset, $\{ead\} \prec c$.

Definition 15. Given an itemset $Y = \{I_{k-1}, \ldots, I_2, I_1\}$, the CoUTlist of its superset $X = \{I_k, I_{k-1}, \ldots, I_2, I_1\}$ is constructed as follows: $\forall\ Y_{\text{node}}$ *in* CoUTlist(Y), if $I_k \in$ the prefixPath of Y_{node}, add Y_{node} to the CoUTlist(X) such that

(i) nodeNo = nodeNo of Y_{node}
(ii) nodeSupp = nodeSupp of Y_{node}
(iii) nodeUtility = nodeUtility + prefixPath$[I_k]$ of Y_{node}
(iv) prefixPath = {key: value for key, value in the prefixPath of Y_{node}, if key $\in X \prec I_k$}

Figure 8 shows how the CoUTlist of $\{da\}$ itemset and CoUTlist of $\{ca\}$ itemset are constructed from the CoUTlist of $\{a\}$ itemset. It further shows how the CoUTlist of $\{c\ da\}$ itemset is constructed from the CoUTlist of $\{da\}$ itemset. The CoUTlist(a) is [[1, 6, 51, 'd': 15, 'c': 72], [2, 2, 24, 'c': 28]]. As item $\{d\}$ has only appeared in the first element of CoUTlist(a), only one element is added to the CoUTlist(da), whose nodeNo = 1, nodeSupp = 6, nodeUtility = 51 + 15 = 66, and prefixPath = c: 72. On the other hand, item $\{c\}$ has appeared in the two elements of CoUTlist(a); hence, two elements are added to the CoUTlist(ca). The first element is with nodeNo = 1, nodeSupp = 6, nodeUtility = 51 + 72 = 123, and prefixPath = { }, while the second element is with nodeNo = 2, nodeSupp = 2, nodeUtility = 24 + 28 = 52, and prefixPath = { }. Similarly, the CoUTlist(cda) is constructed from the CoUTlist(da). As item $\{c\}$ has appeared in the element of CoUTlist(da), this element is added to the CoUTlist(cda) with nodeNo = 1, nodeSupp = 6, nodeUtility = 66 + 72 = 138, and prefixPath = { }.

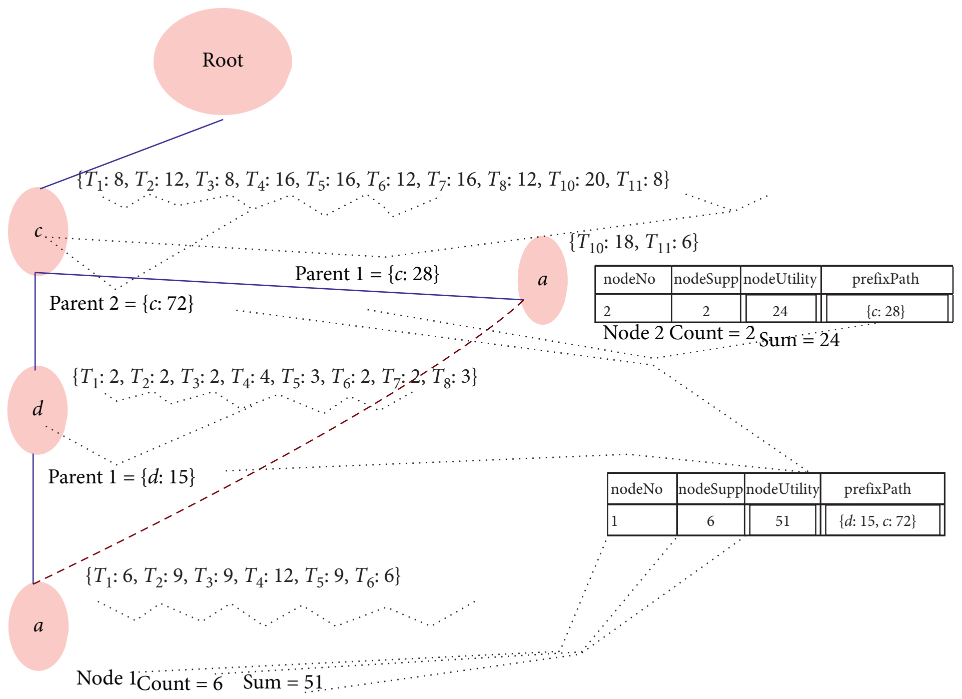

Root
{T1: 8, T2: 12, T3: 8, T4: 16, T5: 16, T6: 12, T7: 16, T8: 12, T10: 20, T11: 8}
c
{T10: 18, T11: 6}
Parent 1 = {c: 28}
a
nodeNo nodeSupp nodeUtility prefixPath
2 2 24 {c: 28}
Node 2 Count = 2 Sum = 24
Parent 2 = {c: 72}
{T1: 2, T2: 2, T3: 2, T4: 4, T5: 3, T6: 2, T7: 2, T8: 3}
d
Parent 1 = {d: 15}
nodeNo nodeSupp nodeUtility prefixPath
1 6 51 {d: 15, c: 72}
{T1: 6, T2: 9, T3: 9, T4: 12, T5: 9, T6: 6}
a
Node 1 Count = 6 Sum = 51

FIGURE 5: CoUTlist of 1-itemset construction.

nodeNo	nodeSupp	nodeUtility	prefixPath
1	6	51	{d: 15, c: 72}
2	2	24	{c: 28}

FIGURE 6: CoUTlist of item {a}.

The CoUTlist of {f} itemset			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	1	40	{e: 60, a: 12, d: 4, c: 16}
2	2	28	{a: 15, d: 5, c: 28}

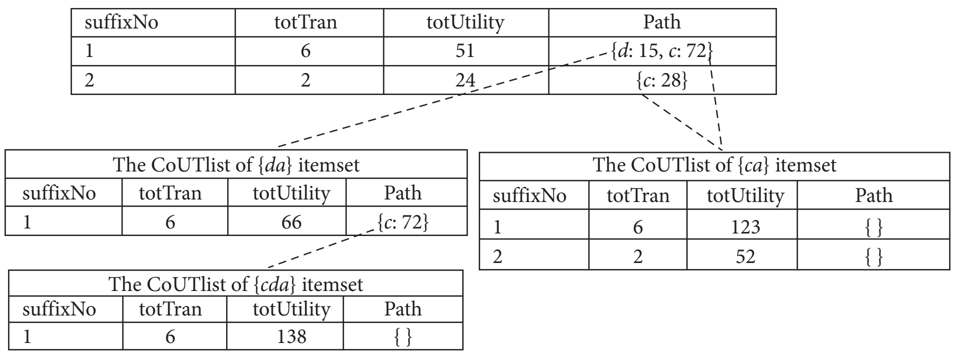
The CoUTlist of {b} itemset			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	2	32	{a: 15, d: 4, c: 20}
2	2	28	{d: 5, c: 28}

The CoUTlist of {e} itemset			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	1	30	{b: 24, a: 6, d: 2, c: 8}
2	2	75	{a: 21, d: 6, c: 24}
3	1	25	{b: 20, d: 2, c: 12}

The CoUTlist of {d} itemset			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	8	20	{c: 100}
2	1	3	{ }

The CoUTlist of {c} itemset			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	10	128	{ }

FIGURE 7: The CoUTlist of the remaining 1-itemsets.



suffixNo	totTran	totUtility	Path
1	6	51	{d: 15, c: 72}
2	2	24	{c: 28}

The CoUTlist of {da} itemset

suffixNo	totTran	totUtility	Path
1	6	66	{c: 72}

The CoUTlist of {ca} itemset

suffixNo	totTran	totUtility	Path
1	6	123	{ }
2	2	52	{ }

The CoUTlist of {cda} itemset

suffixNo	totTran	totUtility	Path
1	6	138	{ }

FIGURE 8: Constructing the CoUTlist of k-itemset.

4.4. Pruning Properties. Alongside adopting Property 1 (TWU) [16], three new pruning properties are introduced in the proposed ECoHUPM algorithm and applied to reduce the search space.

Property 1. Transaction-Weighted Upper Bound Property (TWU) is already introduced in [16]. The proposed algorithm utilizes the TWU property [16] to remove all 1-itemsets having TWU less than minUtil threshold.

Let X be a k-itemset, and let Y be a $(k-1)$-itemset such that $Y \subset X$. If X is HTWUI, Y is HTWUI as well. This means that if an itemset is Low Transaction-Weighted Utilization Itemset (LTWUI), all its supersets will be LTWUIs as well. Hence, this property can be used to reduce the search space by removing LTWUIs with their supersets from the search space.

Property 2. The first proposed pruning property is Upper Bound property based on summation of Utility and the Path Utilities (UBUPU).

Given an itemset Y, if $u(Y)$ plus $pu(Y)$ is less than the minimum utility threshold (minUtil), Y and any superset of Y are not CoHUIs.

Proof. Let X be a superset of Y; we know the following.

The utility of Y is calculated as the sum of the nodes' utilities in the CoUTlist(Y).

$$u(Y) = \sum_{Y_{\text{node}} \in \text{CoUTlist}(Y)} \text{nodeUtility}, \quad (2)$$

for example, $u(a) = 51 + 24 = 75$.

The path utility of Y itemset is calculated as the sum of the prefixes' utilities stored in the prefixpath in the CoUTlist(Y).

$$pu(Y) = \sum_{Y_{\text{node}} \in \text{CoUTlist}(Y)} \text{sum}(\text{prefixPath.values}), \quad (3)$$

for example, $pu(a) = [(15+72)+(28)] = 115$.

Since $Y \subseteq X$,

$$U(X) \leq u(Y) + pu(Y), \quad (4)$$

for example, $u(da) = 66$, $u(ca) = 175$, and $u(c\ da) = 138$. All values $\leq u(a) + pu(a) = 75 + 115 = 190$. □

Property 3. The second proposed pruning property is Lower Bound property based on the Node Utility (LBNU). As the CoUTlist of each itemset X is a list of elements (nodes), where each element represents a set of transactions containing X, on the contrary to the utility list [18, 22] or projected database [20, 24], where each element represents a single transaction, with CoUTlist, there is a possibility that the utility of some itemsets exceeds the min Util in some elements of their CoUTlists, and thus the following lower bound property based on the nodeUtility is formed.

Consider an itemset X: $\forall\ X_{\text{node}} \in \text{CoUTlist}(X)$, if nodeUtility $\geq$ min Util, then all possible combinations of itemsets in the current path are considered high utility itemsets (Figure 9).

Proof. Let CoUTlist(X) be the correlation utility list of itemset X, and $\forall\ X_{\text{node}} \in \text{CoUTlist}(X)$, we know the following:

The set of parent nodes in the current path is denoted as $[P_1, P_2, \ldots, P_n]$.

$u(P_1 + X) = \text{nodeUtility} + \text{prefixPath}[P_1]$.

Hence, $u(i - \text{itemset} + X) \geq \text{nodeUtility} \geq \text{min Util}$, where $i \in [1, n]$).

For example, for the running example, with $m$min Util = 90, as shown in Figure 9, in the CoUTlist(ae), the nodeUtility of the second element = 96 ≥min Util. Hence, all possible combinations of itemsets in the current path {$da\ e$}, {cae}, and {$c\ da\ e$} are considered high utility itemsets. □

Property 4. The third proposed pruning property is Sorted-Reversing Downward Closure (SRDC) property based on Kulc measure, which is used as a correlation measure in the proposed ECoHUPM algorithm. By adopting reverse depth-first traversal, each k-itemset is in this form $\{I_k, I_{k-1}, \ldots, I_2, I_1\}$, and because these items are sorted based on their support descending order, the sorted-reversing property based on Kulc measure is formed as

$$\text{kulc}(I_k I_{k-1} I_{k-2}, \ldots, I_1) \leq \text{kulc}(I_{k-1} I_{k-2}, \ldots, I_1). \quad (5)$$

The CoUTlist (ae)			
nodeNo	nodeSupp	nodeUtility	prefixPath
1	1	36	{*d*: 2, *c*: 8}
2	2	96	{*d*: 6, *c*: 24}

Figure 9: The CoUTlist of {ae} itemset.

Proof. Let $X = \{I_k, I_{k-1}, \ldots, I_2, I_1\}$ be a superset of $Y = \{I_{k-1}, \ldots, I_2, I_1\}$; we know that

$$\begin{gathered}\sup(X) \leq \sup(Y),\\ \sup(I_{k-1}) \leq \sup(I_k).\end{gathered} \tag{6}$$

Hence,

$$\begin{aligned}\operatorname{Kulc}(X) &= \frac{1}{k}\left[\frac{\sup(X)}{\sup(I_k)} + \frac{\sup(X)}{\sup(I_{k-1})} + \cdots + \frac{\sup(X)}{\sup(I_1)}\right]\\ \leq \operatorname{Kulc}(Y) &= \frac{1}{k-1}\left[\frac{\sup(Y)}{\sup(I_{k-1})} + \frac{\sup(Y)}{\sup(I_{k-2})} + \cdots + \frac{\sup(XY)}{\sup(I_1)}\right].\end{aligned} \tag{7}$$

Note that the proposed SRDC property is similar to the sorted downward closure (SDC) property which was employed in [20, 24]. However, SDC cannot be applied directly in the proposed ECoHUPM algorithm, because, in [20, 24], the items are sorted in the ascending order of their supports. Meanwhile, in ECoHUPM, the items are sorted in the descending order of their supports.

The proposed UBUPU and SRDC properties are employed to reduce the search space by removing all supersets of each Y itemset if $\operatorname{kulc}(Y) < (\min\operatorname{Corr})$ or $\operatorname{sum}(u(Y), pu(Y)) < \min\operatorname{Util}$. On the other hand, the proposed LBNU property is employed to improve the searching efficiency as follows: in each element in the CoUTlist of Y itemset, if nodeUtility is equal to or greater than min Util, all possible supersets of Y in the current path are considered as HUIs. Hence, ECoHUPM needs only to calculate the correlation of each superset X in the current path without needing to make sure that $u(X)$ or $\operatorname{sum}(u(X) + pu(X))$ exceeds min Util.

Figure 10 shows how these three properties help significantly in reducing the search space and thus improve the mining performance. For example, with $\min\operatorname{Corr} = 0.4$ and $\min\operatorname{Util} = 90$, since $\operatorname{kulc}(ef) = 0.29 < \min\operatorname{Corr}$, $\{ef\}$ and all its supersets are removed from the search space based on the proposed SRDC property. Similarly, the itemsets $\{bf\}$, $\{abe\}$, and $\{ab\}$ are removed along with their supersets from the search space according to the proposed UBUPU and SRDC properties. As nodeUtility of the second element of CoUTlist(ae) is greater than min Util, all its supersets in the current path $\{da\ e\}$, $\{cae\}$, and $\{c\ da\ e\}$ are considered HUIs as stated in the proposed LBNU property, and thus ECoHUPM only needs to calculate their *Kulc* values to examine whether they are Correlated HUIs or not (Algorithm 1). □

4.5. ECoHUPM Algorithm. Using the UTtree and the CoUTlist, we developed an efficient algorithm named ECoHUPM for mining Correlated High Utility Itemsets. ECoHUPM adopts reverse depth-first traversal for set-enumeration tree searching and employs the pruning properties to reduce the search space. The pseudocode of ECoHUPM is shown in Algorithm 2.

The input for ECoHUPM is database D including transactional database with external utility along with min Util, as a given minimum utility threshold, and min Corr, as a given minimum correlation threshold. In line 1, the ECoHUPM preprocess database D to obtain the revised database RD, and then it stores the set of unique items sorted in the descending order of their support in itemsList (line 2). Then it runs Algorithm 1 to construct the UTtree by performing one scan of the RD (line 3).

Lines 4 to 15 state the procedure of extracting the Correlated High Utility Itemsets. For each loop started by line 4, ECoHUPM finds all Correlated HUIs that are supersets of item X. Lines 5 and 6 construct the CoUTlist of 1-itemset X with the help of interLink and parentLink of nodes whose label is X in the UTtree as is illustrated in Section 4.3.1. As the correlation value of each 1-itemset is 1, lines 7 to 9 add X itemset to the CoHUPs list if its utility is equal or greater than min Util. Line 10 employs the proposed UBUPU property by examining the summation of utility and path utilities of an itemset X. If the sum $(u(X) + pu(X))$ is less than min Util, all its supersets will be pruned using the proposed UBUPU property. Otherwise, the function search is called to search its supersets (line 11). This procedure is recursively performed for all 1-itemsets to discover Correlated High Utility Itemsets (Algorithm 3).

The function $\operatorname{search}(X, \operatorname{CoUTlist}(X), \min\operatorname{Util}, \min$ Corr) is used to search the whole list of extensions of itemset X in order to discover all correlated high utility supersets. It scans $\operatorname{CoUTlist}(X)$ node by node to find all possible prefixes (lines 1–12). If the utility of the current node nodUtility is equal to or greater than min Util, all $prefixes$ in the prefixPath of the current node are added to the list of high utility prefixes HUprefixesList (lines 5 to 7). All unique prefixes are added to the prefixList (lines 8 to10). Then the procedure in lines 13 to 29 is performed for each prefix P_i in prefixList. First, 1-extension itemset of X is formed such that itemset $= P_i + X$ (line 14) and its CoUTlist is constructed (line 15). Line 16 implements the proposed SRDC property to remove the itemset with all its supersets from the search space if its Kulc value is less than min Corr. Lines 17 to 19 employ the proposed LBNU property to add an itemset to the list of CoHUPs if current P_i is in the HUprefixList and search function is called to search all its supersets. Otherwise, lines 21 to 23 add an itemset to the list of CoHUPs if its utility is equal to or greater than min Util, while lines 24–26 employ the proposed UBUPU property to remove the itemset with all its supersets if its utility plus path utilities is less than min Util; otherwise, search function is called to search its supersets.

5. Experiment Design

In this section, we present the design of the experiments for performance evaluation. Experiments were performed on a

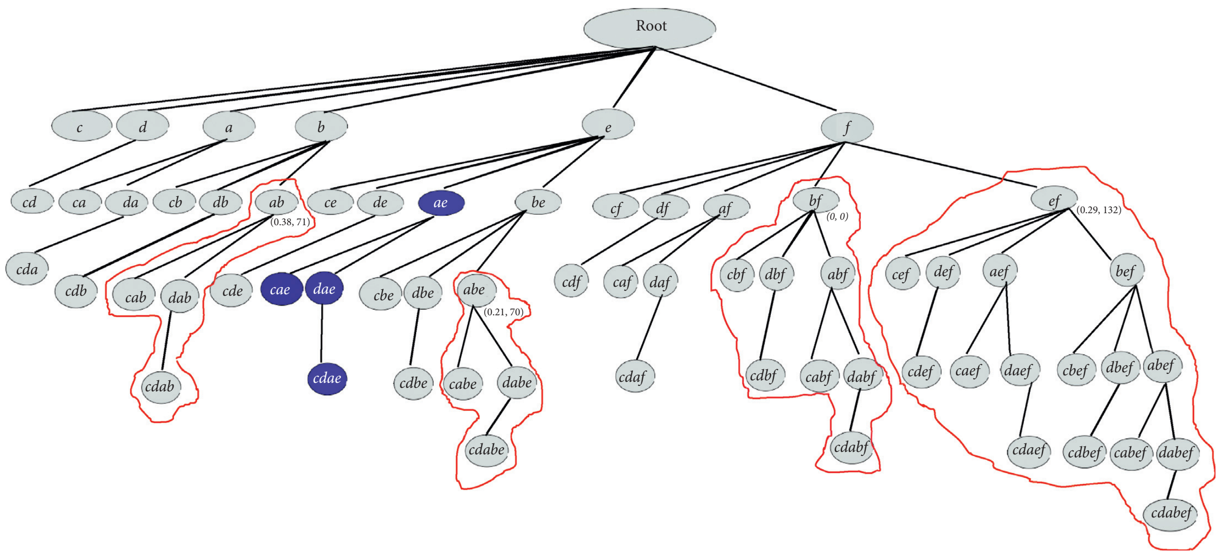


Figure 10: Pruning search space.

```
        Input: The revised Database RD
        Output: The UTtree
        Initialization: Create Root node to initialize the tree
 (1)    for each transaction $T_i$ in RD do
 (2)        itemsList← Items of $T_i$
 (3)        Call insertTree ($T_i$, itemsList, Root)
 (4)    end for
 (5)    return UTtree
        Function: insertTree ($T_i$, itemsList, $N\ d$)
 (1)    firstItem←itemsList[0]
 (2)    if $N\ d$ has $N$child such that $N$.label = firstItem then
 (3)        Update the utList of $N$
 (4)    else
 (5)        Create new node $N$ for the firstItem
 (6)        $N$.parentLink←$N\ d$
 (7)        $N$.interLink points to the nodes whose label is firstItem
 (8)        remainItems←itemsList[1: ]
 (9)        if remainItems is not null then
(10)            Call insertTree ($T_i$, remainItems, $N$)
(11)        end if
(12)    end if
```

Algorithm 1: Algorithm for constructing UTtree.

computer with an Intel® Core$^{\text{TM}}$ i7-6600U CPU @ 2.60 GHz (4 CPUs), 2.8 GHz, and 8 GB of memory, running 64-bit Windows 10 Pro. The performance of the proposed ECoHUPM algorithm was compared to that of the CoHUIM and CoHUI-Miner algorithms in terms of both runtime and memory consumption. All algorithms were implemented using Python 3 and Jupyter Notebook.

5.1. Datasets Used. We used five standard datasets downloaded from SPMF library [54], two real-life datasets with real utility values (Foodmart and Ecommerce), and three real-life datasets with synthetic utility values (BMS, Chess, and Mushroom). Characteristics of the considered datasets are shown in Table 6.

min Corr is adapted with three times on each sparse dataset and four times on each dense dataset to evaluate the efficiency of the proposed ECoHUPM algorithm and they are denoted, respectively, as ECoHUPM$_{\text{-minCorr1}}$, ECoHUPM$_{\text{-minCorr2}}$, ECoHUPM$_{\text{-minCorr3}}$, and ECoHUPM$_{\text{-minCorr4}}$. The different three min Corr thresholds are, respectively, set as follows: (1) in Foodmart dataset, 0.1, 0.2, 0.3; (2) in Ecommerce dataset, 0.1, 0.2, 0.3; (3) in BMS dataset, 0.5, 0.6, 0.7; (4) in Mushroom dataset, 0.4, 0.45, 0.5, 0.55; and (5) in Chess dataset, 0.7, 0.75, 0.8, 0.85. Because all algorithms use the same measures to evaluate the

```
     Input: Database $D$, min Util, and min Corr.
     Output: All Correlated High Utility Itemsets.
(1)  Scan $D$ to obtain the revised database RD;
(2)  itemsList← the set of unique items in RD sorted on their support descending order;
(3)  Execute the Algorithm 1 to construct UTtree for the RD;
(4)  while itemsList not NULL do
(5)    $X$← the last item in itemsList;
(6)    from the header table of the UTtree follow the interLink of the nodes labeled $X$ to construct the CoUTlist$(X)$;
(7)    if $u(X) \geq$ min Util then
(8)      CoHUPs←CoHUPs $\cup X$
(9)    end if
(10)   if $u(X) + pu(X) \geq$ min Util then
(11)     Call search ($X$, CoUTlist$(X)$, min Util, min Corr)
(12)   end if
(13)   itemsList←itemsList- $[X]$
(14) end while
(15) return CoHUPs
```

Algorithm 2: ECoHUPM algorithm.

```
     Function: search ($X$, CoUTlist$(X)$, min Util, min Corr)
(1)  HUprefixList←$\varnothing$
(2)  prefixList←$\varnothing$
(3)  for each $X_{\text{node}} \in$ CoUTlist$(X)$ do
(4)    for each $P_i \in$ prefixPath do
(5)      if nodeUtility $\geq$ min Util then
(6)        HUprefixList←HUprefixList $\cup P_i$
(7)      end if
(8)      if $P_i \notin$ prefixList then
(9)        prefixList←prefixList $\cup P_i$
(10)     end if
(11)   end for
(12) end for
(13) for each $P_i \in$ prefixList do
(14)   itemset←$P_i + X$;
(15)   Scan the CoUTlis$(X)$ to construct the CoUTlist(itemset)
(16)   if Kulc(itemset) $\geq$ min Corr then
(17)     if $P_i \in$ HUprefixList then
(18)       CoHUPs←CoHUPs $\cup$ itemset
(19)       Call search(itemset, CoUTlist(itemset), min Util, min Corr)
(20)     else
(21)       if $u$(itemset) $\geq$ min Util then
(22)         CoHUPs←CoHUPs $\cup$ itemset
(23)       end if
(24)       if $u$(itemset) + $pu$(itemset) $\geq$ min Util then
(25)         Call search (itemset, CoUTlist(itemset), min Util, min Corr)
(26)       end if
(27)     end if
(28)   end if
(29) end for
```

Algorithm 3: Algorithm for searching the list of extensions of itemset X.

interestingness of the extracted patterns, all algorithms resulted in the same number of patterns in all experiments. Tables 7–11 show the total number of Correlated HUIs when min Util and min Corr are varied in each dataset.

5.2. Runtime. The runtime of the proposed ECoHUPM algorithm was compared with those of two state-of-the-art Correlated HUIs mining algorithms: CoHUIM and CoHUI-Miner. For each dataset, the min Util threshold was adjusted,

Table 6: Characteristics of the used datasets.

Dataset name	Description	Transactions count	Items count	Average items count per transaction	Has real utility values?	Sparse or dense
Foodmart	Dataset of customer transactions from a retail store, obtained and transformed from SQL-Server 2000.	4,141	1,559	4.42	Yes	Sparse dataset
Ecommerce	Transactional dataset which contains all the transactions occurring between 01/12/2010 and 09/12/2011 of a UK-based and registered nonstore online retail.	14,975	3,803	15.4	Yes	Sparse dataset
BMS	Click stream data from a web store used in KDD-Cup 2000.	59,602	497	2.51	No	Very sparse dataset
Mushroom	Prepared based on the UCI mushrooms dataset.	8,124	119	23	No	Dense dataset
Chess	Prepared based on the UCI chess dataset.	3,196	75	37	No	Very dense dataset

Table 7: The total number of Correlated HUIs when min Util and min Corr are varied in Foodmart dataset.

min Util	0.00001	0.00005	0.0001	0.0005	0.001
min Corr 1	53443	52776	47510	2102	258
min Corr 2	1951	1942	1893	925	257
min Corr 3	1575	1573	1564	915	257

Table 8: The total number of Correlated HUIs when min Util and min Corr are varied in Ecommerce dataset.

min Util	0.0005	0.0007	0.0009	0.0011	0.0013
min Corr 1	4956	3040	1857	1167	799
min Corr 2	1345	977	745	592	467
min Corr 3	901	645	479	386	318

Table 9: The total number of Correlated HUIs when min Util and min Corr are varied in BMS dataset.

min Util	0.00001	0.00005	0.0001	0.0005	0.001
min Corr 1	4255	1233	788	518	231
min Corr 2	930	510	388	271	190
min Corr 3	539	411	335	251	188

Table 10: The total number of Correlated HUIs when min Util and min Corr are varied in Mushroom dataset.

min Util	0.005	0.006	0.007	0.008	0.009
min Corr 1	95559	93034	89771	86276	82922
min Corr 2	48099	47118	45759	44473	43380
min Corr 3	24041	23633	23070	22645	22372

Table 11: The total number of Correlated HUIs when min Util and min Corr are varied in Chess dataset.

min Util	0.005	0.006	0.007	0.008	0.009
min Corr 1	119279	119268	119261	119250	119203
min Corr 2	49256	49245	49238	49234	49231
min Corr 3	19485	19475	19469	19465	19463

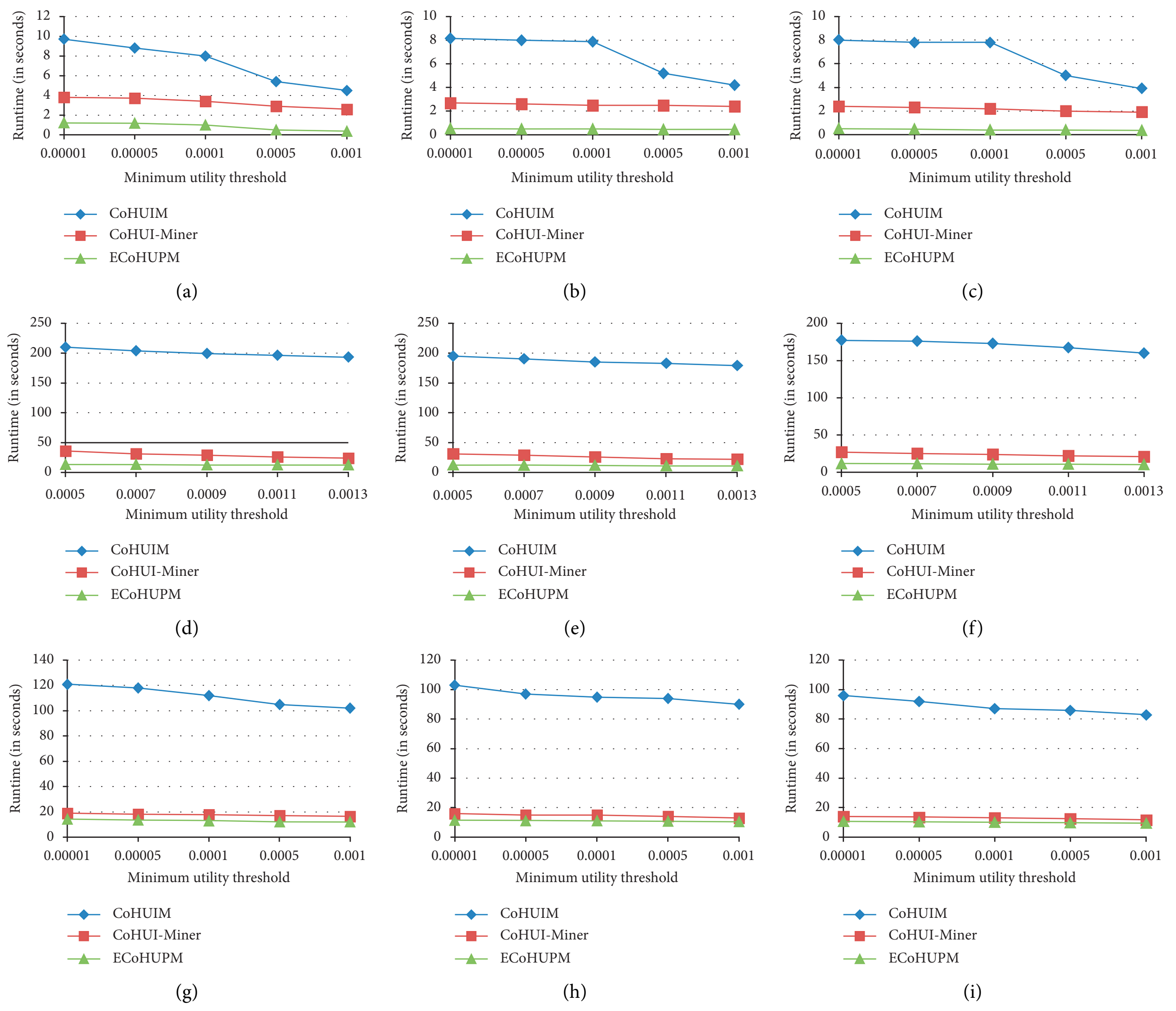


FIGURE 11: Runtime for sparse datasets. (a) Foodmart (minCorr = 0.1). (b) Foodmart (minCorr = 0.2). (c) Foodmart (minCorr = 0.3). (d) Ecommerce (minCorr = 0.1). (e) Ecommerce (minCorr = 0.2). (f) Ecommerce (minCorr = 0.3). (g) BMS (minCorr = 0.5). (h) BMS (minCorr = 0.6). (i) BMS (minCorr = 0.7).

and the runtime execution of each algorithm was calculated. Figures 11 and 12 show the results for the sparse and dense datasets, respectively. It can be observed that ECoHUPM is faster than CoHUIM and CoHUI-Miner. Further, it is found that ECoHUPM is significantly faster than CoHUIM on sparse datasets such as Foodmart (up to 12.8 times) and Ecommerce (up to 15.6 times) and on very sparse datasets such as BMS (up to 8.7 times). For the dense datasets, it is observed that, in most cases, the CoHUIM fails to discover the Correlated HUIs when the min Util and min Corr thresholds are set to low values as is shown on the Mushroom dataset with min Corr = [0.35–0.45] and min Util = [0.005–0.009]. Meanwhile, when the thresholds are set to high values such as min Corr = 0.5 and min Uti = [0.05–0.09], the ECoHUPM is faster than CoHUIM (up to four times). Similarly, on very dense datasets such as Chess, CoHUIM fails to discover the Correlated HUIs with min Corr = [0.7–0.8] and min Util = [0.005–0.009]. Meanwhile, when the thresholds are set to high values such as min Corr = 85 and min Uti = [0.08–0.28], the ECoHUPM is faster than CoHUIM (up to 4.6 times).

On the other hand, the ECoHUPM is faster than the CoHUI-Miner on sparse datasets such as Foodmart and Ecommerce (up to 5 and 2.2 times, respectively) and it is slightly faster on very sparse dataset such as BMS (up to 1.3 times). For dense datasets with low threshold values, the ECoHUPM is significantly faster than the CoHUI-Miner on the Mushroom dataset (up to 3.2 times) and on very dense datasets such as Chess (up to 4.3 times faster). Meanwhile, with high threshold values, the ECoHUPM is slightly faster than the CoHUI-Miner on Mushroom dataset (up to 2.1 times faster) and on Chess dataset (up to 1.4 times faster).

The main reason why the ECoHUPM algorithm is always faster than CoHUIM and CoHUI-Miner algorithms is that the novel CoUTlist structure is highly effective in reducing the database size as compared to the projection mechanism used on CoHUIM and CoHUI-Miner algorithms. That is, in

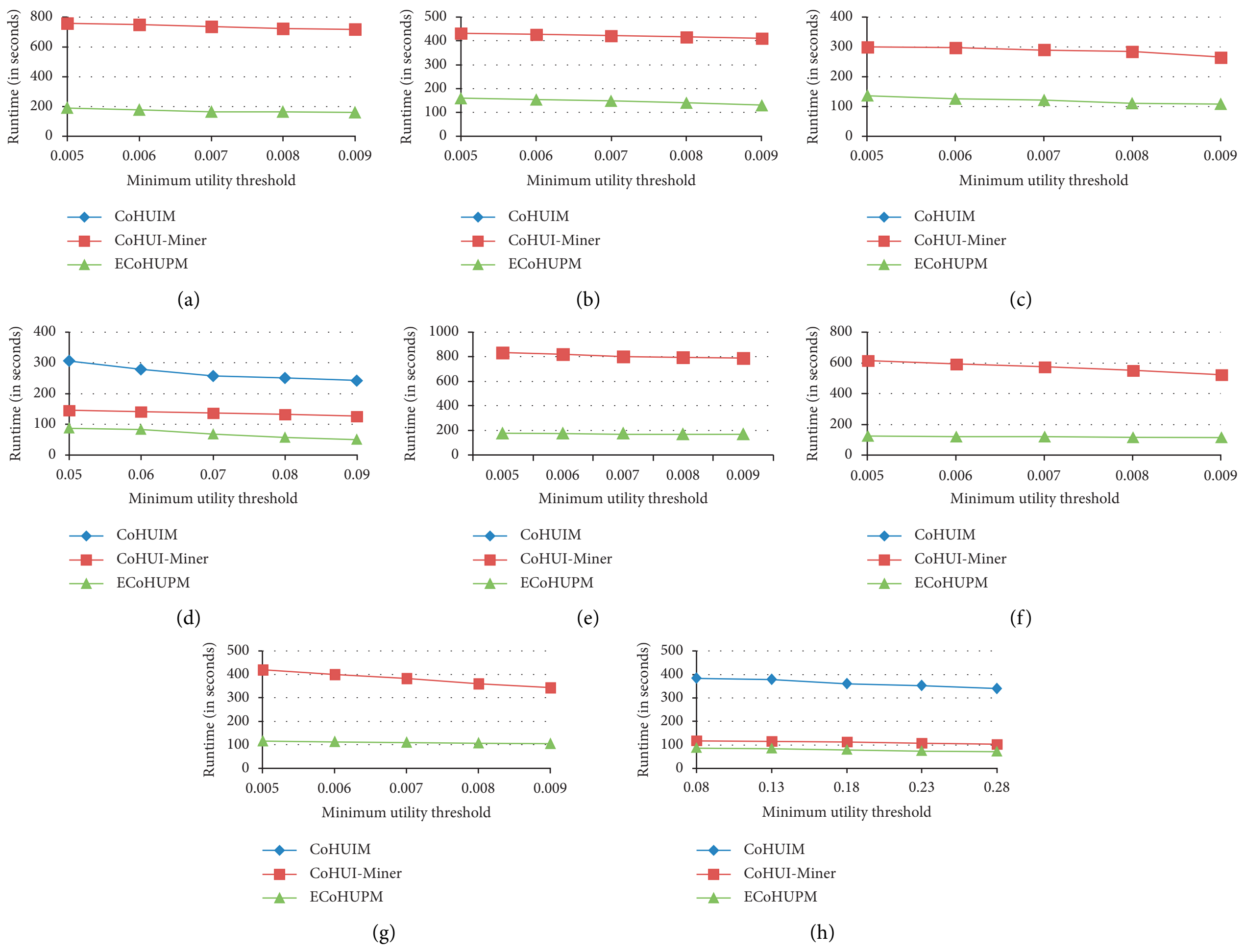


Figure 12: Runtime for dense datasets. (a) Mushroom (minCorr = 0.35). (b) Mushroom (minCorr = 0.4). (c) Mushroom (minCorr = 0.45). (d) Mushroom (minCorr = 0.5). (e) Chess (minCorr = 0.70). (f) Chess (minCorr = 0.75). (g) Chess (minCorr = 0.80). (h) Chess (minCorr = 0.85).

the CoUTlist, each element represents a set of transactions where the itemset occurs in the same path. Meanwhile, in the projected database, each element represents a single transaction where the itemset occurs. Moreover, the proposed pruning properties help in reducing the search space.

The CoHUIM algorithm performs two phases. It first generates the candidate itemsets whose correlation is equal to or greater than the min Corr threshold, and then it calculates the utility of each candidate. Hence, in all datasets, CoHUIM is much slower than the CoHUI-Miner and the proposed ECoHUPM.

The size of the projected database of an itemset increases as the density of the datasets is increased and thus the cost of building the projected database of the supersets is also increased. Thus, the CoHUIM could not find the Correlated HUIs when it was run on Mushroom and Chess datasets with low min Util and min Corr thresholds. This is because it suffers from excessive dataset scanning in the second phase.

The CoHUI-Miner is a One-Phase algorithm. However, due to the big size of the projected database of each itemset as compared to the CoUTlist especially in dense and very dense datasets, the proposed ECoHUPM is significantly faster than the CoHUI-Miner on Mushroom and Chess datasets.

In very sparse datasets, the size of the CoUTlist of each itemset is slightly smaller than the size of the projected database. Hence, the proposed ECoHUPM is slightly faster than the CoHUI-Miner in BMS dataset.

5.3. Memory Usage.

The comparison of the memory usage of the proposed ECoHUPM against CoHUIM and CoHUI-Miner is shown in Figure 13. In this figure, the *Y*-axis represents the memory usage which is measured by the memory_usage module in Python.

It is observed that the proposed ECoHUPM algorithm consumes less memory as compared to the CoHUIM and CoHUI-Miner in all datasets. More specifically, on the sparse datasets such as Foodmart and Ecommerce, the memory usage of the CoHUIM occupies 1.5 and 3 times the memory of the proposed ECoHUPM, respectively. Meanwhile, on very sparse datasets such as BMS, the memory usage of the CoHUIM occupies 2.2 times the memory of the proposed ECoHUPM.

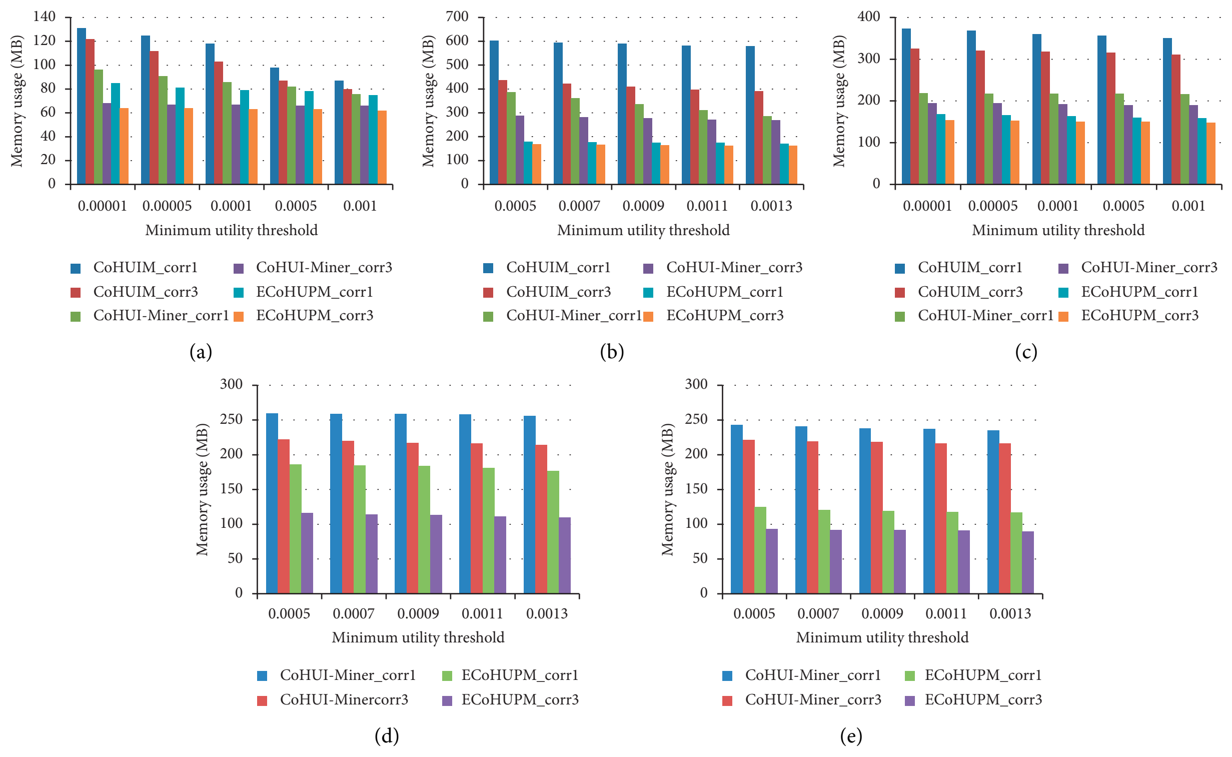


FIGURE 13: Memory usage. (a) Foodmart. (b) Ecommerce. (c) BMS. (d) Mushroom. (e) Chess.

Likewise, on sparse and very sparse datasets such as Foodmart, Ecommerce, and BMS, the CoHUI-Miner occupies 1.07, 1.8, and 1.3 times the memory of the proposed ECoHUPM, respectively. Meanwhile, on dense and very dense datasets such as Mushroom and Chess, the CoHUI-Miner occupies 1.7 and 2.2 times the memory of the proposed ECoHUPM, respectively.

6. Conclusion

This paper proposed an efficient algorithm named ECoHUPM for mining Correlated HUIs. The ECoHUPM algorithm adopts divide-and-conquer approach and employs UTtree structure which is an extended form of FP-tree. A novel data structure based on the UTtree named CoUTlist is proposed in the ECoHUPM to store sufficient information for mining the desired patterns in an efficient manner. Three new pruning properties have been introduced and applied to reduce the search space and improve the mining performance. The first proposed pruning property is Upper Bound property based on summation of Utility and the Path Utilities (UBUPU), the second one is Lower Bound property based on the Node Utility (LBNU), and the third one is Sorted-Reversing Downward Closure (SRDC) property based on Kulc measure.

An extensive experimental evaluation was conducted on five datasets including sparse, very sparse, dense, and very dense datasets. The experimental results show that the proposed ECoHUPM algorithm is efficient as compared to the state-of-the-art CoHUIM and CoHUI-Miner algorithms in terms of both time and memory consumption.

Data Availability

The data used in the experiments of this paper are available at SPMF library [54]: https://www.philippe-fournier-viger.com/spmf/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] J. Desjardins, *How Much Data is Generated Each Day?*, https://www.visualcapitalist.com/how-much-data-is generated-each-day/, 2019.

[2] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts Techniques*, Elsevier, Amsterdam, The Netherlands, 2011.

[3] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *WIREs Data Mining Knowledge Discovery*, vol. 7, no. 4, Article ID e1207, 2017.

[4] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[5] C.-H. Chee, J. Jaafar, I. A. Aziz, M. H. Hasan, and W. Yeoh, "Algorithms for frequent itemset mining: a literature review," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2603–2621, 2019.

[6] J.-F. Qu, M. Liu, and P. Fournier-Viger, "Efficient algorithms for high utility itemset mining without candidate generation," in *Studies in Big Data*, pp. 131–160, Springer, Cham, Switzerland, 2019.

[7] C. C. Aggarwal, M. A. Bhuiyan, and M. A. Hasan, "Frequent pattern mining algorithms: a survey," in *Frequent Pattern Mining*, pp. 19–64, Springer, Cham, Switzerland, 2014.

[8] C. C. Aggarwal, "Applications of frequent pattern mining," in *Frequent Pattern Mining*, pp. 443–467, Springer, Cham, Switzerland, 2014.

[9] S. Naulaerts, P. Meysman, W. Bittremieux et al., "A primer to frequent itemset mining for bioinformatics," *Briefings in Bioinformatics*, vol. 16, no. 2, pp. 216–231, 2015.

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

[11] C. F. Ahmed, S. K. Tanbeer, B.-S. Byeong-Soo Jeong, and Y.-K. Young-Koo Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708–1721, 2009.

[12] Q.-H. Duong, B. Liao, P. Fournier-Viger, and T.-L. Dam, "An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies," *Knowledge-Based Systems*, vol. 104, pp. 106–122, 2016.

[13] B. Vo, L. T. T. Nguyen, N. Bui, T. D. D. Nguyen, V.-N. Huynh, and T.-P. Hong, "An efficient method for mining closed potential high-utility itemsets," *IEEE Access*, vol. 8, pp. 31813–31822, 2020.

[14] J. M.-T. Wu, J. C.-W. Lin, and A. Tamrakar, "High-utility itemset mining with effective pruning strategies," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 6, pp. 1–22, 2019.

[15] X. Han, X. Liu, J. Li, and H. Gao, "Efficient top-k high utility itemset mining on massive data," *Information Sciences*, vol. 557, pp. 382–406, 2020.

[16] Y. Liu, W.-K. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," *Advances in Knowledge Discovery and Data Mining*, Springer, Berlin, Germany, pp. 689–695, 2005.

[17] S. Dawar and V. Goyal, "UP-Hist tree: an efficient data structure for mining high utility patterns from transaction databases," in *Proceedings of the 19th International Database Engineering & Applications Symposium*, pp. 56–61, Yokohama, Japan, 2015.

[18] W. Gan, J. Chun-Wei, H.-C. Chao, T.-P. Hong, and S. Y. Philip, "CoUPM: correlated utility-based pattern mining," in *Proceedings of the IEEE International Conference on Big Data (Big Data)*, pp. 2607–2616, IEEE, Seattle, WA, USA, 2018.

[19] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and H.-J. Choi, "A framework for mining interesting high utility patterns with a strong frequency affinity," *Information Sciences*, vol. 181, no. 21, pp. 4878–4894, 2011.

[20] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and H. Fujita, "Extracting non-redundant correlated purchase behaviors by utility measure," *Knowledge-Based Systems*, vol. 143, pp. 30–41, 2018.

[21] J. C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, and H.-C. Chao, "FDHUP: fast algorithm for mining discriminative high utility patterns," *Knowledge and Information Systems*, vol. 51, no. 3, pp. 873–909, 2017.

[22] P. Fournier-Viger, Y. Zhang, J. C.-W. Lin, D.-T. Dinh, and H. Bac Le, "Mining correlated high-utility itemsets using various measures," *Logic Journal of the IGPL*, vol. 28, no. 1, pp. 19–32, 2020.

[23] W. Gan, J. C.-W. Lin, H.-C. Chao, H. Fujita, and P. S. Yu, "Correlated utility-based pattern mining," *Information Sciences*, vol. 504, pp. 470–486, 2019.

[24] B. Vo, L. V. Nguyen, V. V. Vu et al., "Mining correlated high utility itemsets in one phase," *IEEE Access*, vol. 8, pp. 90465–90477, 2020.

[25] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM Sigmod Record*, vol. 29, no. 2, pp. 1–12, 2000.

[26] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 482–486, SIAM, Lake Buena Vista, FL, USA, 2004.

[27] 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, no. 6, pp. 7419–7424, 2011.

[28] W. Song, Y. Liu, and J. Li, "Mining high utility itemsets by dynamically pruning the tree structure," *Applied Intelligence*, vol. 40, no. 1, pp. 29–43, 2014.

[29] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pp. 55–64, Maui, HI, USA, 2012.

[30] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning," *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 83–92, 2014.

[31] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[32] Y. Chen and A. An, "Approximate parallel high utility itemset mining," *Big Data Research*, vol. 6, pp. 26–42, 2016.

[33] Q.-H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørvåg, and T.-L. Dam, "Efficient high utility itemset mining using buffered utility-lists," *Applied Intelligence*, vol. 48, no. 7, pp. 1859–1877, 2018.

[34] M. K. Sohrabi, "An efficient projection-based method for high utility itemset mining using a novel pruning approach on the utility matrix," *Knowledge and Information Systems*, vol. 62, no. 11, pp. 4141–4167, 2020.

[35] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and S. Y. Philip, "Efficient algorithms for mining top-k high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 54–67, 2015.

[36] R. Gunawan, E. Winarko, and R. Pulungan, "A BPSO-based method for high-utility itemset mining without minimum utility threshold," *Knowledge-Based Systems*, vol. 190, Article ID 105164, 2020.

[37] P. Fournier-Viger, J. C.-W. Lin, T. Gueniche, and P. Barhate, "Efficient incremental high utility itemset mining," in *Proceedings of the ASE Big Data & Social Informatics*, pp. 1–6, Kaohsiung, Taiwan, 2015.

[38] U. Yun and H. Ryang, "Incremental high utility pattern mining with static and dynamic databases," *Applied Intelligence*, vol. 42, no. 2, pp. 323–352, 2015.

[39] G.-C. Lan, T.-P. Hong, J.-P. Huang, and V. S. Tseng, "On-shelf utility mining with negative item values," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3450–3459, 2014.

[40] G.-C. Lan, T.-P. Hong, and V. S. Tseng, "Discovery of high utility itemsets from on-shelf time periods of products," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5851–5857, 2011.

[41] T.-L. Dam, K. Li, P. Fournier-Viger, and Q.-H. Duong, "An efficient algorithm for mining top-k on-shelf high utility itemsets," *Knowledge and Information Systems*, vol. 52, no. 3, pp. 621–655, 2017.

[42] B.-E. Shie, P. S. Yu, and V. S. Tseng, "Efficient algorithms for mining maximal high utility itemsets from data streams with different models," *Expert Systems with Applications*, vol. 39, no. 17, pp. 12947–12960, 2012.

[43] C. Manike and H. Om, "Modified GUIDE (LM) algorithm for mining maximal high utility patterns from data streams," *International Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 517–529, 2015.

[44] T.-L. Dam, K. Li, P. Fournier-Viger, and Q.-H. Duong, "CLS-Miner: efficient and effective closed high-utility itemset mining," *Frontiers of Computer Science*, vol. 13, no. 2, pp. 357–381, 2019.

[45] C.-W. Wu, P. Fournier-Viger, J.-Y. Gu, and V. S. Tseng, "Mining closed+ high utility itemsets without candidate generation," in *Proceedings of the 2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 187–194, IEEE, Tainan, Taiwan, 2015.

[46] J. C.-W. Lin, Y. Djenouri, G. Srivastava, U. Yun, and P. Fournier-Viger, "A predictive GA-based model for closed high-utility itemset mining," *Applied Soft Computing*, vol. 108, Article ID 107422, 2021.

[47] J. C.-W. Lin, Y. Djenouri, and G. Srivastava, "Efficient closed high-utility pattern fusion model in large-scale databases," *Information Fusion*, vol. 76, pp. 122–132, 2021.

[48] E. R. Omiecinski, "Alternative interest measures for mining associations in databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 1, pp. 57–69, 2003.

[49] S. Bouasker and S. Ben Yahia, "Key correlation mining by simultaneous monotone and anti-monotone constraints checking," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pp. 851–856, New York, NY, USA, 2015.

[50] M. Barsky, S. Kim, T. Weninger, and J. Han, "Mining flipping correlations from large datasets with taxonomies," 2011, https://arxiv.org/abs/1201.0233.

[51] T. Wu, Y. Chen, and J. Han, "Re-examination of interestingness measures in pattern mining: a unified framework," *Data Mining and Knowledge Discovery*, vol. 21, no. 3, pp. 371–397, 2010.

[52] N. B. Younes, T. Hamrouni, and S. B. Yahia, "Bridging conjunctive and disjunctive search spaces for mining a new concise and exact representation of correlated patterns," in *International Conference on Discovery Science*, pp. 189–204, Springer, Berlin, Germany, 2010.

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

[54] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, "SPMF: a java open-source pattern mining library," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3389–3393, 2014.