# An Efficient Approach for High Utility Itemset Mining

**Ph.D. Synopsis**

Submitted To

**Gujarat Technological University**

For the Degree

of

**Doctor of Philosophy**

in

**Computer / IT Engineering**

By

**Patel Sureshkumar Bhikhabhai**

Enrollment No: 189999913029

**Supervisor:**

**Dr. Sanjay M. Shah**

**Professor & Head CE Department,**

**Government Engineering College Rajkot, Gujarat, India**

# Table of Contents

# 1. Title of the Thesis and Abstract

## 1.1. Title of the Thesis

An Efficient Approach for High Utility Itemset Mining.

## 1.2. Abstract

The conventional approach for association rule mining is a frequent itemset mining. It is widely used and popular for extracting related items. Traditional approach focuses on whether the group of items is frequently appear in the dataset or not. However, in certain real-world scenarios, it becomes essential to consider the quantity and importance of items. For example, in a supermarket, identifying profitable items from customer transaction data, or in the medical field, discovering combinations of symptoms that are highly indicative of diseases. High utility itemset mining incorporates item's quantity and importance to addresses this need. Numerous research studies have been conducted on high utility itemset mining, with utility list based methods emerging as efficient techniques. These methods avoid the generation of candidate sets, which can be computationally expensive. However, a major drawback of existing utility list based techniques is the need for costly join operations on utility lists. These operations can degrade the algorithm's performance by increasing execution time and storage requirements. The cost of the utility list join operations is directly related to the number of comparisons required to find out the common transactions between the utility lists. In this research, proposed SCAO (Support Count Ascending Order) based search space exploration technique to reduce the number of comparisons required to find common transactions between the utility lists. So, it will minimize the cost of the join operations. Also, existing state of the art approaches perform unnecessary utility list join operations of the itemsets which are not high utility itemsets. To deal with this problem, proposed PUCP (Predicted Utility Co-exist Pruning) uses PUCS (Predicted Utility Co-exist Structure) to eliminate unnecessary join operations. The proposed approach using the PUCP called as PUCP-Miner. The performance of the proposed approaches are evaluated with the existing algorithms like HUI-Miner, mHUI-Miner and ULB miner on some of the standard real datasets. The experimental results demonstrate that the proposed SCAO-based approach and PUCP-miner have outperformed existing state-of-the-art methods by up to 59% in terms of execution time and up to 46% in memory consumption.

# 2. A brief description of the problem and state of the art approaches

Data mining techniques, including Frequent Itemset Mining (FIM) [2] and High Utility Itemset Mining (HUIM)[3][4][5], are utilized to uncover crucial patterns concealed within vast datasets[1]. FIM is extensively applied in real-life scenarios such as customer behaviour analysis, examining the contribution of symptoms to predict diseases, and identifying valuable customers. However, a significant limitation of FIM is its sole consideration of item present or absent within transactions [2][6]. To overcome this limitation, it is essential to incorporate item quantities and their importance, which play a vital role in various applications, especially in transaction databases [3] [7] [8] [9]. High Utility Itemset Mining (HUIM) focuses on identifying itemsets that generate significant profits or have high importance [3] [5] [10] [11] [12]. This technique takes into account both the count and profit of the items in the itemset. HUIM extracts sets of items whose utility exceeds a user-defined threshold. The HUIM problem originates from FIM, but it presents greater challenges compared to FIM due to the lack of adherence to the downward closure property. In FIM, the apriori characteristic is utilized to effectively reduce the search space [2]. In contrast to FIM, utility measures employed in HUIM cannot be directly utilized for efficient search space reduction. This is primarily due to the fact that subsets of high utility itemsets may have low utility, and conversely, subsets of low utility itemsets may have high utility. Consequently, the reduction of the search space becomes more challenging in HUIM compared to FIM. In the context of transaction databases, several methods have been developed to discover high utility itemsets, employing different data structures and pruning strategies. HUIM approaches are mainly categorised as candidate generation and test based, Tree based and Utility list based. Among them Utility list based approach is most recent and efficient approach. List based approaches does not generate the candidate itemsets [14]. Despite the better performance of these approaches, costly utility list join operations limits the performance. The efficiency of join operations in the utility list based approach depends on the number of comparisons required to identify the common transactions in both utility lists. This comparison process influences the cost of join operations and consequently, impacts the overall performance and running time of the algorithm.

## 2.1 Problem Background

Let set I = {$i_1$, $i_2$, $i_3$ ………,$i_n$} single itemset. Database (DB) consist of set of transactions and utility table as shown in Table1 and Table2, respectively. The utility table defines the utility/profit of an individual item. The transaction consist of set of items with quantity. The set of transactions are ($T_1$, $T_2$, $T_3$……$T_m$). Each transaction is uniquely identified as $T_i$. The transaction $T_i$ is defined as

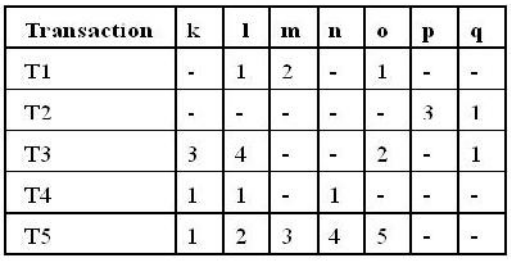
$$T_i = \{ i_k : q( i_k) \mid 1 \leq i \leq m, 1 \leq k \leq n\} \tag{1}$$

The set of items in the transaction $T_i$ is subset of I. The set of items in $T_i$ are associated with quantities defined as count q($i_k$) (where 1 ≤ k ≤ n) considered as an internal utility of item in the transaction. The utility table maintains the utility (importance / weight / profit) values p(i) of each item i in I, which is considered as an external utility.

**Definition 1: Item's Utility.**

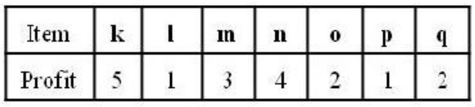
For the item $i_k$ in the transaction $T_j$, the item's utility of $i_k$ is u ($i_k$,$T_j$) = q($i_k$,$T_j$) × p($i_k$). i.e u(a,$T_3$)= q (m, $T_5$) × p(m) = 15

Table 1: Sample Transaction Database



| Transaction | k | l | m | n | o | p | q |
|---|---|---|---|---|---|---|---|
| T1 | - | 1 | 2 | - | 1 | - | - |
| T2 | - | - | - | - | - | 3 | 1 |
| T3 | 3 | 4 | - | - | 2 | - | 1 |
| T4 | 1 | 1 | - | 1 | - | - | - |
| T5 | 1 | 2 | 3 | 4 | 5 | - | - |

Table 2: Sample Utility in DB



| Item | k | l | m | n | o | p | q |
|---|---|---|---|---|---|---|---|
| Profit | 5 | 1 | 3 | 4 | 2 | 1 | 2 |

**Definition 2: Itemset's Utility.**

For the itemset Px in the transaction $T_j$, the itemset Px's utility is

$$u(Px,Tj) = \sum_{ik \in P x \wedge ik \in Ti} u(ik,Tj) \tag{2}$$

i.e consider itemset Px={m,n}, $u(Px,T_5)=u(m,T_5) + u(n,T_5)=25$

**Definition 3: Itemset's utility in database DB.**

For the itemset Px and the transaction database DB, the Px's utility in DB is

$$u(Px) \quad = \sum_{ik \subset I \wedge Px \in Ti} u(Px,Tj) \tag{3}$$

i.e consider itemset Px={k,l}, $u(Px)=u(Px,T_3) + u(Px,T_4) + u(Px,T_5) = 32$

**Definition 4: High Utility itemset.**

The itemset Px is high utility itemset, if it's utility value is higher than the user defined *MinUtility* threshold

$$\text{HUIset} = \{Px \mid Px \subseteq I,\ u(Px) \geq \mathit{MinUtility}\} \tag{4}$$

i.e consider *MinUtility*=15 and itemset Px = {k,l}, u(k,l) is 32 that is more than *MinUtility* indicating that Px is a high utility itemset.

**Definition 5: High Utility Itemset Mining.**

Find out all the itemsets from the transaction database, which has utility value more than the user specified minimum utility (*MinUtility*) threshold.

**Definition 6: Transaction's Utility**

The transaction's utility is obtained by summing up the utility values of all the items present in that particular transaction. It is defined as

$$TU(Ti) = \sum_{\forall ik \in Ti} u(ik,Ti) \tag{5}$$

i.e $TU(T_1) = u(l,T_1) + u(m,T_1) + u(n,T_1) = 9$

**Definition 7: Transaction Weighted Utility (TWU)**

The Transaction Weighted Utility (TWU) of an itemset Px is calculated as the sum of the transaction utilities of all transactions that contain the itemset Px.

$$TWU(Px) = \sum_{Px \in Ti} TU(Ti) \tag{6}$$

i.e TWU(n) is the total of transaction utility of $T_4$ and $T_5$ as n belongs to Transaction $T_4$ and $T_5$, So TWU(n) is 52

## 2.2 The traditional approaches for High Utility Itemset Mining

Based on different mining techniques, data structure and pruning strategies, HUIM algorithms are mainly classified into three categories

(1) Candidate generation and test based approaches

(2) Tree based approaches

(3) Utility List based approaches.

### 2.2.1 Candidate generation and test based approaches for High Utility Itemset Mining

Apriori based approaches generates level wise candidate sets and prune some itemsets based on various measures like TWU, Utility upper bound, expected utility etc[1][4][5][9][16][27]. The main drawback of these approaches are, it scans the database multiple times. It generates the huge amount of candidate itemsets as it relies on the loose upper bound for pruning. It also generates some of the patterns which are not present in the database resulting waste of time to process these patterns, making algorithms less efficient.

### 2.2.2 Tree based approaches for High Utility Itemset Mining

Tree based approaches for HUIM work in three steps:

(1) Scan the database one or more time and construct the tree.

(2) Restructure the tree to reduce the candidate sets, decrease the overestimation etc.

(3) Discover the high utility mining from the restructured tree.

Compress the dataset into tree structure. Utility measures like TWU, sum of item quantity used for the pruning which are overestimation [7][8][12][19]. These are the pattern growth approaches. For restructure/mining it recursively process the tree that is time consuming task.

### 2.2.3 Utility List based approaches for High Utility Itemset Mining

Previous approaches for high utility itemset mining generates candidate itemsets. It is the time consuming and consume more memory to store the candidate itemsets. In 2012, Liu and Qu proposed first single phase algorithm for high utility itemset mining without candidate generation [14]. It reduces the mining time as it removes the huge amount of candidate generation problem of previous apriori based and tree based approaches. It proposed novel data structure utility list to store the utility information of the itemsets and efficient pruning strategies based on sum of item utility and the remaining utility. It maintains the utility lists of itemsets.  Novel Utility list structure is a triplet <*Tid, iutility, rutility*> where *Tid* represents the transaction ID in which the itemset exist, *iutility* is the utility value of the itemset, *rutility*  is the heuristic information  which stores sum of the utilities of items that come after itemset in the transaction. It explores the search space as TWU in ascending order to gradually extend the itemset. It performs the costly join operations on utility list of (k-1) itemsets to construct the utility list of k-itemset. It also constructs the unnecessary utility list of the itemsets which are not available in the dataset. In 2017, Peng, Koh and Riddle proposed tree structure [13] to avoid the generation of utility lists of the itemsets which are not present in the database. The HUI Miner and FHM[18] explore the search space using the set enumeration tree so these algorithms construct the utility list of some itemsets which are not exist in database. Due to that, HUI Miner and FHM algorithms are somewhat inefficient. Modified HUIMiner incorporates tree structure IHUP into HUIMiner. As per IHUP Tree Structure property, the path of tree corresponds to transaction in database means information about all the items in a transaction is stored together in the tree. In 2017, Duong, Viger, Ramampiaro, Norvag and Dam proposed efficient high utility itemset mining using buffered utility list [17]. Authors proposed novel list structure called utility list buffer to store the item utility information and efficient join operation to generate a segment of itemsets in linear time. The ULB is based on the buffering utility information to reduce the memory consumption. The ULB structure reuses the memory of the itemset that will not further expanded. In 2019, Qu, Liu and Viger proposed new structure Utility-List* [10] which is higher in performance than HUI-Miner. It has been observed that in HUI-Miner the utility list of k-itemsets is constructed using the Tid comparisons of both utility lists of (k-1) itemsets but all the Tid comparisons are not effective. The effective comparisons are on matching of the Tid. The ineffective comparison degrades the algorithm performance. The HUI-Miner* remove these ineffective comparison by Utiliiy-list* structure. The utility list based approaches are performing better in terms of execution time and memory consumption. Although the performance of these approaches are limited due to unnecessary costly utility list join operations.

## 2.3 Open Issue

The current methods for HUIM suffer from time consuming operations and high memory requirements due to generation of large number of candidate itemsets, inefficient pruning mechanisms. These approaches employ multiple pruning measures that tend to overestimate, resulting in unnecessary computations and increased resource usage. Utility list based approaches outperform in HUIM as they do not generate candidate itemsets. However, the performance of list based algorithms is limited by the need to perform a large number of expensive utility list join operations. These join operations contribute to the computational overhead and can affect the efficiency of the algorithms. Therefore, there is a need to address the computational cost associated with utility list join operations to further enhance the performance of list based HUIM algorithms. The cost of join operations in utility list based approaches is directly proportional to the number of comparisons needed to find common transactions between the utility lists. Join count, number of comparisons are the challenges to reduce the cost of the utility list join operations. Hence, it will improve the performance of the mining algorithm in execution time and memory consumption.

## 3. Objective, Scope of the work, and Problem Statement

### 3.1 Aim and Research Objectives

The main aim of this research is to develop efficient high utility itemset mining approach by eliminating the unnecessary utility list join operations and reducing the number of comparisons. This research work proposes to achieve the following objectives:

- To study and investigate existing methods for the high utility itemset mining.
- To identify the challenges for the high utility itemset mining.
- To identify the scope to improve the performance of the high utility itemset mining methods.
- To develop and investigate the efficient search space exploration technique to reduce the cost of utility list join operations by reducing the number of comparisons required to join utility lists.
- To design novel structure to store the predicted utility of the itemsets that can be used to develop efficient pruning mechanism.
- To develop and investigate efficient pruning mechanism to reduce the number of join operations by eliminating unnecessary join operations of utility lists.
- To evaluate performance of proposed approaches and compare the results with existing state-of-the-art methods.

### 3.2 Scope of the research

The aim of this research is, improve the performance of High Utility Itemset Mining approaches. The research is mainly focused on transactional datasets.

### 3.3 Problem Statement

The problem statement of this research is:

*"To Design an Efficient High Utility Itemset Mining Approach to eliminate unnecessary join operations and decrease the number of comparisons for utility list join operations"*

## 4. Research contribution

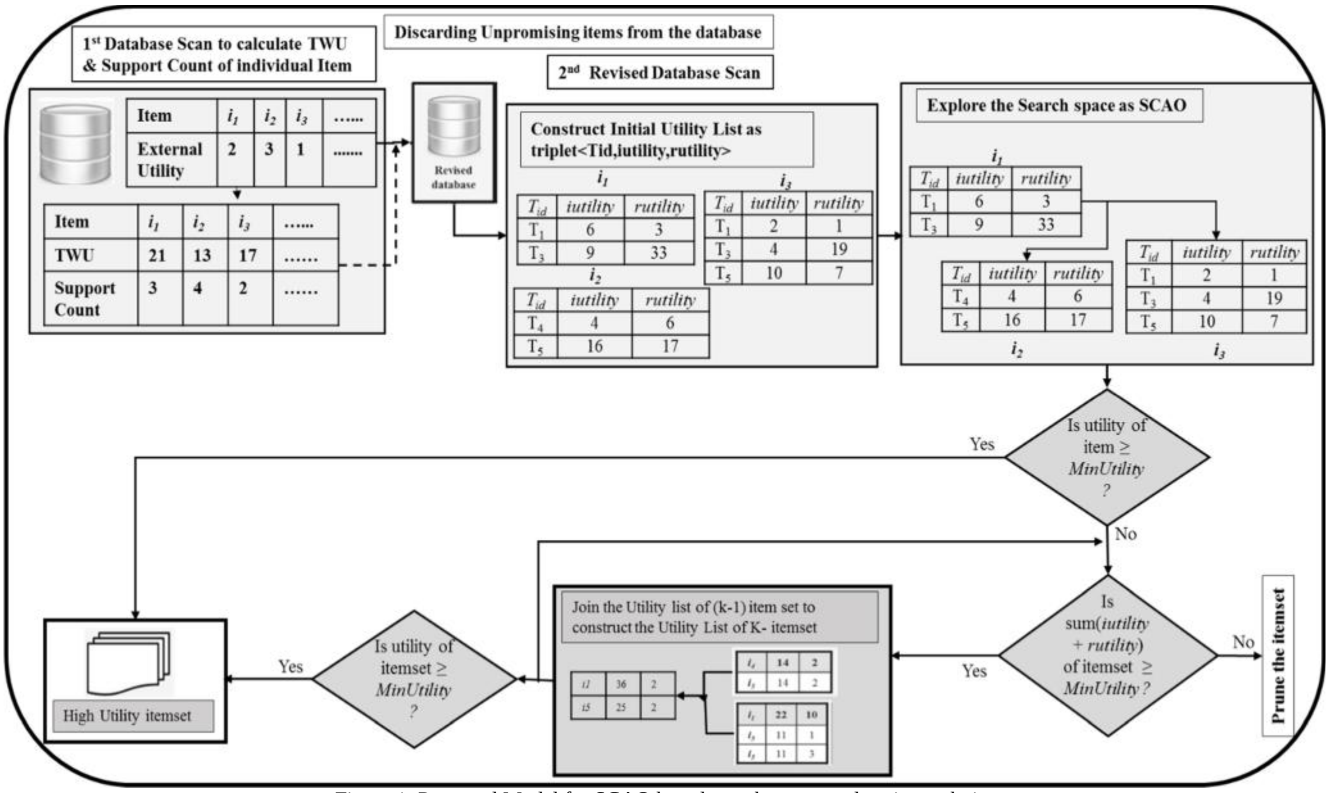
The main contribution of this research is to design efficient utility list join operations by reducing the number of comparisons by efficiently exploring the search space required to find out the common elements between the utility lists. To design an efficient pruning mechanism PUCP (Predicted Utility Co-exist pruning) to reduce the number of join operations.

## The methodology of research

The research methodology comprises the developing a novel search space exploration sequence to minimize the cost associated with utility list join operations. Proposed PUCP approach reduces the number of join operations. Performances of the proposed approaches evaluated with state of art methods on standard real datasets.

### 4.1 An Efficient search space exploration technique for high utility itemset mining

Exploration of the search space is the order in which the itemset is extended. The search space is maintained as set enumeration tree as shown in Figure 2. The existing HUIM approaches explore the search space as TWU ascending order to extend the itemsets. The proposed approach explores the search space as support count ascending order that can reduce the number of comparisons required to join the utility lists. The proposed SCAO-based HUIM process model is presented in Figure 1. The proposed approach works in three phases.



1st Database Scan to calculate TWU & Support Count of individual Item
Discarding Unpromising items from the database
2nd Revised Database Scan
Revised database
Construct Initial Utility List as triplet<Tid,iutility,rutility>
Explore the Search space as SCAO
Is utility of item ≥ MinUtility ?
Yes
No
Is sum(iutility + rutility) of itemset ≥ MinUtility ?
Yes
No
Prune the itemset
Join the Utility list of (k-1) item set to construct the Utility List of K- itemset
Is utility of itemset ≥ MinUtility ?
Yes
High Utility itemset

| Item | $i_1$ | $i_2$ | $i_3$ | …… |
|---|---|---|---|---|
| External Utility | 2 | 3 | 1 | ……. |

| Item | $i_1$ | $i_2$ | $i_3$ | …… |
|---|---|---|---|---|
| TWU | 21 | 13 | 17 | …… |
| Support Count | 3 | 4 | 2 | …… |

$i_1$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_1$ | 6 | 3 |
| $T_3$ | 9 | 33 |

$i_2$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_4$ | 4 | 6 |
| $T_5$ | 16 | 17 |

$i_3$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_1$ | 2 | 1 |
| $T_3$ | 4 | 19 |
| $T_5$ | 10 | 7 |

|   |   |   |
|---|---|---|
| $i_4$ | 14 | 2 |
| $i_5$ | 14 | 2 |

|   |   |   |
|---|---|---|
| $i_1$ | 22 | 10 |
| $i_3$ | 11 | 1 |
| $i_5$ | 11 | 3 |

|   |   |   |
|---|---|---|
| i1 | 36 | 2 |
| i5 | 25 | 2 |

Figure 1: Proposed Model for SCAO based search space exploration technique.

**Phase 1: Construction of a revised database**

Scan the dataset to calculate the TWU and support count for each item. Unpromising items, which have a TWU lower than the *MinUtility* threshold, are then discarded. The database is scanned once again to rearrange all transactions in ascending order based on their support count. The resulting set of revised transactions is referred as the revised database. For instance, the data in Table 1 and 2, with *MinUtility* value is 40, Items p and q are discarded as TWU of items fall below the threshold. The items in each transaction are then rearranged in ascending order based on their support count, resulting in revised transactions such as m-n-o-k-l.

**Phase 2: Construction of initial utility list**

The database is scanned again to create the initial utility list for each promising item. The utility list of items consists of triplets associated with the transaction containing the item. Each triplet contains the following information: TRid (Item's transaction ID), *iutility* (Item's utility value in a revised transaction), and *rutility* (Item's remaining utility value) [12, 13, 14, 15, 16].

**Definition 8:** For the itemset X and transaction T, all the items in Transaction T that come after itemset X where X ⊆ T is denoted as T|X. i.e T5 | mn = {okl} and T3 | o = {kl}

Table 3: Utility Lists of 1-Itemset and 2-itemset

{m}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| $T_1$ | 6 | 3 |
| $T_5$ | 9 | 33 |

{n}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| $T_4$ | 4 | 6 |
| $T_5$ | 16 | 17 |

{o}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| $T_1$ | 2 | 1 |
| $T_3$ | 4 | 19 |
| $T_5$ | 10 | 7 |

{k}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| $T_3$ | 15 | 4 |
| $T_4$ | 5 | 1 |
| $T_5$ | 5 | 2 |

{l}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| $T_1$ | 1 | 0 |
| $T_3$ | 4 | 0 |
| $T_4$ | 1 | 0 |
| $T_5$ | 2 | 0 |

{mn}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| T5 | 25 | 17 |

{mo}

| Tid | *Iutility* | *rutility* |
|---|---|---|
| T1 | 8 | 1 |
| T5 | 19 | 7 |

**Definition 9: Remaining utility**

The total utility of all the items that follows the itemset X in transaction T, is represented as *rutility*(X,T).

$$rutility(X,T) = \sum_{i\in(T|X)} u(i,T) \text{ where } X\subseteq T \quad (7)$$

For example, constructing the item o's utility list in transaction $T_3$, *iutility* value of o in $T_3$ is 4 and remaining utility *rutility* value of o in $T_3$ = *rutility* (o, $T_3$) = u( k, $T_3$) + u(l, $T_3$) = 15+4 =19. Similarly, the initial utility list of all promising items (m, n, o, k, l) is constructed as shown in Table 3.

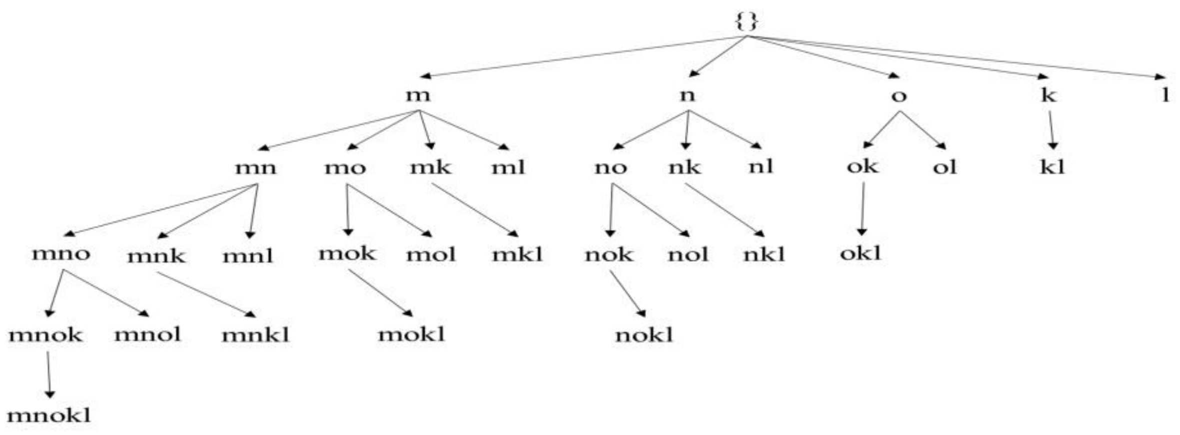


Figure 2: Set Enumeration Tree

**Phase 3: Search space exploration & mining process**

The search space can be represented by a set enumeration tree represented in Figure 2. Once the initial utility list is created, the proposed technique explores the search space in ascending order of support count called Support Count Ascending Order based search space exploration technique (SCAO). It recursively extends the (k-1) itemset by combining with its successor item. Pruning is performed based on the sum of *iutility* and *rutility* to determine whether the itemset can be further extended. The specific details of the pruning mechanism is in section 4.1.2. The utility lists of the k-itemset are constructed by joining the utility lists of the (k-1) itemset and utility list of its successor item. The details of the utility list join operation is in section 4.1.1. Additionally, the technique simultaneously discovers the high utility itemsets.

### 4.1.1 A Construction of Utility list of 2-itemset and k-itemset

Consider the 2-itemset x = {mn}. To construct the utility list of itemset x, the join operation is performed on the utility lists of m and n. During the join operation, a common transaction is searched from both utility lists, and the element <*Tid*, *iutility*, *rutility*> is added to the utility list of {mn}. Here, *Tid* represents the common transaction ID, *iutility* represents the sum of *iutility* values from utility lists of {m} and {n}, and *rutility* represents the *rutility* value of {n}, as itemset {n} comes after itemset {m} according to SCAO. For example, when constructing the utility list of {mn}, the utility lists of m and n are joined. There is a common transaction ID $T_5$, the element <$T_5$, 25, 17> is added to utility list of {mn}. Similarly, add all the elements which have common transactions into the utility list of {mn}. Table 3 shows the utility lists of 2-itemsets. The utility list of a k-itemset can be constructed by joining the utility lists of two (k-1) itemsets. By performing a join operation, the common transactions between the two (k-1) itemsets are identified, and the corresponding elements are added to the utility list of the k-itemset. This process allows for the construction of utility lists for higher-order itemsets by leveraging the utility information from smaller itemsets.

### 4.1.2 Pruning Mechanism

In depth search space exploration identified all high utility itemset, but it consumes more time because large number of items are available in datasets. Therefore, it is necessary to trim the search space by discarding the itemsets that does not contribute to the high utility. Use the itemset *iutility* and *rutility* values from the itemset's utility list to narrow the search field. Only those itemsets are extended further if total of its *iutility* and *rutility* values is no less than *MinUtility* threshold. Otherwise it can be discarded as per lemma-1[16] because any of the superset of such itemset is not a high utility itemset. The total of *iutility* values from itemset's utilitylist is the utility of the itemset. The itemset is a high utility itemset if it's utility is no less than the *MinUtility* criterion.

**Lemma-1**:- If the sum of all *iutility* and *rutility* values from X's utility list UL(X) is no less than the *minUtility*, then any itemset X' that is the extension of itemset X is not a high utility itemset.

i.e, consider the itemset {mn}'s utility list $UL_{mn}$, the total of itemset's *iutility* and *rutility* is 42, which is larger than the *MinUtility* threshold 40. So it can be further extended. While the total of *iutility* and *rutility* values of {mo} is 35, so it can be discarded without further extended.

### 4.1.3 Analysis of Proposed Method Vs. state of the art methods.

The use of SCAO based search space exploration techniques reduce the number of comparisons required to utility list join operations. An analysis of their time complexity reveals that the proposed SCAO-based algorithms require fewer comparisons when compared to other methods for joining utility lists. Let's consider the utility lists $UL_a$, $UL_b$, and $UL_c$, which represent the utility values of 1-itemsets a, b, and c, respectively. The support counts for these utility lists are denoted as p, q, and r, respectively, with the condition that $p \geq q \geq r$. Now, for the construction of the utility list of itemset ab, perform a join operation between $UL_a$ and $UL_b$.

**Case 1:** Consider the order sequence a-b-c is TWU ascending order of itemset a, b and c. Existing state-of-art algorithms construct the utility lists in sequence as $UL_{ab}$, $UL_{ac}$, $UL_{abc}$. To construct $UL_{ab}$ required $q \log_2 p$ comparisons while $UL_{ac}$ required $r \log_2 p$. The maximum number of entries in $UL_{ab}$ is q and in $UL_{ac}$ is r.

$UL_{abc}$ , Utilitylist of abc constructed by performing join operation on $UL_{ab}$ and $UL_{ac}$, the minimum number of comparisons are $r \log_2 q$. Therefore, the total numbers of comparisons are $q \log_2 p + r \log_2 p + r \log_2 q$

While SCAO-based approach constructs the utility list in the sequence as $UL_{cb}$, $UL_{ca}$, and then $UL_{cba}$ because the SCAO sequence is c-b-a. To construct $UL_{cb}$ required $r \log_2 q$, and to construct $UL_{ca}$ required $r \log_2 p$. The maximum number of entries in $UL_{cb}$ is r, and $UL_{ca}$ is r. To construct the utility list

$UL_{cba}$ of itemset cba by joining $UL_{cb}$ and $UL_{ca}$, the number of comparisons is $r \log_2 r$. Therefore, total number of comparisons are $r \log_2 q + r \log_2 p + r \log_2 r$ which is lesser or equal to $q \log_2 p + r \log_2 p + r \log_2 q$ ($\because r \leq q$ and $r \leq p \Rightarrow r \log_2 r \leq q \log_2 p$).

**Case 2:** TWU ascending order for itemset a, b and c is a-c-b. Existing state-of-art algorithms construct the utility lists in sequence as $UL_{ac}$, $UL_{ab}$, $UL_{acb}$. So, the total numbers of comparisons are $r \log_2 p + q \log_2 p + r \log_2 q$. While SCAO based approach constructs the utility list in the sequence as $UL_{cb}$, $UL_{ca}$, and then $UL_{cba}$, so the total number of comparisons is $r \log_2 q + r \log_2 p + r \log_2 r$, which is lesser or equal then $r \log_2 p + q \log_2 p + r \log_2 q$ ($\because r \leq q$ and $r \leq p \Rightarrow r \log_2 r \leq q \log_2 p$).

**Case 3:** TWU ascending order for itemset a, b and c is b-a-c. Existing state-of-art algorithms construct the utility lists in sequence as $UL_{ba}$, $UL_{bc}$, $UL_{bac}$. So, the total numbers of comparisons are $q \log_2 p + r \log_2 q + r \log_2 q$. While in proposed SCAO based approach required total $r \log_2 q + r \log_2 p + r \log_2 r$ comparisons which is lesser or equal then $q \log_2 p + r \log_2 q + r \log_2 q$ ($\because r \leq q \Rightarrow r \log_2 p \leq q \log_2 p$ and $r \log_2 r \leq r \log_2 q$ ).

**Case 4:** Consider the order sequence b-c-a is TWU ascending order of itemset a, b and c. Existing state-of-art algorithms construct the utility lists in sequence as $UL_{bc}$, $UL_{ba}$, $UL_{bca}$ So, the total numbers of comparisons are $r \log_2 q + q \log_2 p + r \log_2 q$. While Proposed SCAO based approach required total $r \log_2 q + r \log_2 p + r \log_2 r$ comparisons which is lesser or equal then $r \log_2 q + q \log_2 p + r \log_2 q$ ($\because r \leq q \Rightarrow r \log_2 p \leq q \log_2 p$ and $r \log_2 r \leq r \log_2 q$).

**Case 5:** Consider the order sequence c-a-b is TWU ascending order of itemset a, b and c. Existing state-of-art algorithms construct the utility lists in sequence as ULca, ULcb, ULcab. So, the total numbers of comparisons are $r \log_2 p + r \log_2 q + r \log_2 r$., which are the same as the proposed SCAO-based method.

**Case 6:** TWU ascending order for itemset a, b and c is c-b-a. The proposed SCAO-based algorithm performs Utility list join sequence order c-b-a also. Therefore, the numbers of comparisons are the same.

From all the above cases, it has been proved that the proposed join sequence support count ascending order (SCAO) requires fewer comparisons. Hence, it reduces the cost of utility list join operations.

### 4.1.4 Performance Evaluation

The proposed approach SCAO-based search space exploration technique incorporates into existing algorithms such as HUI-Miner, mHUI-Miner, and ULB-Miner. To evaluate the effectiveness of this approach, extensive testing is conducted on diverse real datasets using different *MinUtility* percentages. The experimental results are then compared with other state-of-the-art methods.

#### 4.1.4.1 Experimental Environment

All experimental algorithms were implemented in Java and the tests were conducted on a system with 8GB RAM and an Intel Core i5 processor running Windows 10 Pro. To assess the algorithm's performance under different *MinUtility* values, standard real-time datasets [17] were used. Table 4 provides details of the dataset properties and a comprehensive description. The datasets exhibited variations in the number of items, transactions, and transaction lengths.

Table 4: Characteristics of Dataset [20]

| Sr.No | Dataset Name | Number of Transactions | Number of Items | Average Length |
|---|---|---|---|---|
| 1 | Foodmart | 4141 | 1559 | 4.4 |
| 2 | Connect | 67,557 | 129 | 43 |
| 3 | Chess | 3196 | 75 | 37 |
| 4 | Retail | 88,162 | 16,470 | 10.3 |
| 5 | BMS | 59,602 | 497 | 2.51 |
| 6 | Kosark | 990002 | 41270 | 8.1000 |
| 7 | ecommerce | 14975 | 3468 | 11.71 |

### 4.1.4.2 Performance Evaluation with HUI-Miner

The compared algorithms were executed on diverse datasets using progressively decreasing utility thresholds until either the execution time became excessive or the memory reached its capacity. The execution time was recorded during these experiments. The running time of the proposed SCAO-HUI-Miner is compared with HUI-miner on various real dataset. The result analysis as in Table 5 and Figure 3 shows the proposed SCAO-HUI-Miner outperform with HUI-Miner by 6 to 21 percent on some real dataset.

Table 5: Execution time comparison HUI-Miner Vs SCAO-HUI-Miner

| Dataset | HUI-Miner (Execution time in ms) | SCAO-HUI-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| retail | 617 | 479 | 21.32 |
| bms | 1088 | 904 | 16.86 |
| foodmart | 1105 | 1019 | 8.09 |
| chess | 4139 | 3611 | 12.43 |
| kosark | 4420 | 3748 | 15.16 |

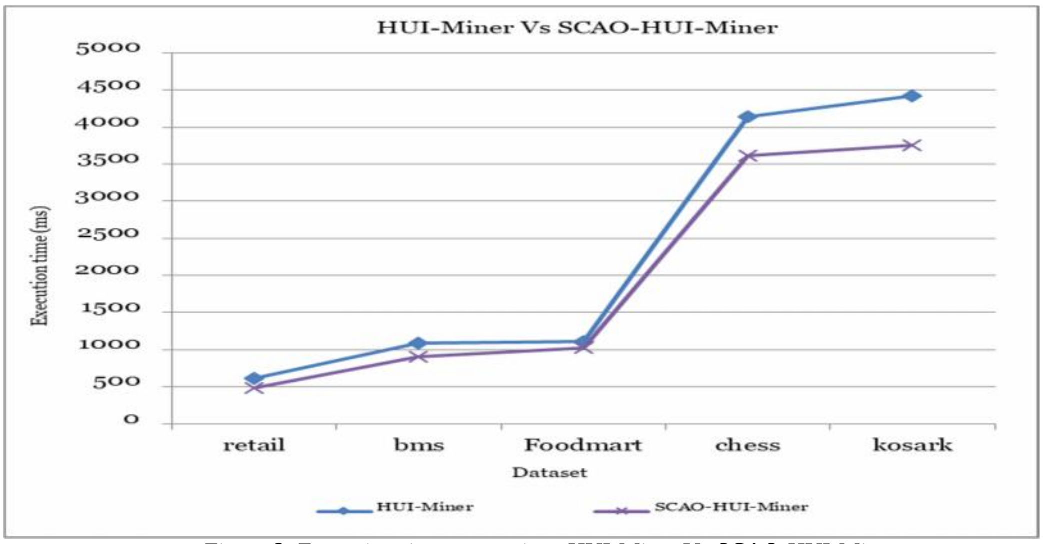


Figure 3: Execution time comparison HUI-Miner Vs SCAO-HUI-Miner

### 4.1.4.3 Performance Evaluation with mHUI-Miner

The comparative analysis of running time required for the proposed approach employs to mHUI-Miner called SCAO-mHUI-Miner and mHUIMiner as shown in Table 6 and Figure 4. Proposed technique SCAO- mHUIMiner take 6% to 23% less time than mHUI-Miner.

Table 6: Execution time comparison mHUI-Miner Vs SCAO-mHUI-Miner

| Dataset | mHUI-Miner (Execution time in ms) | SCAO-mHUI-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| Foodmart | 225 | 177 | 21.24 |
| bms | 385 | 295 | 23.37 |
| retail | 566 | 528 | 6.48 |
| chess | 3067 | 2535 | 18.85 |
| kosark | 5241 | 4770 | 8.55 |

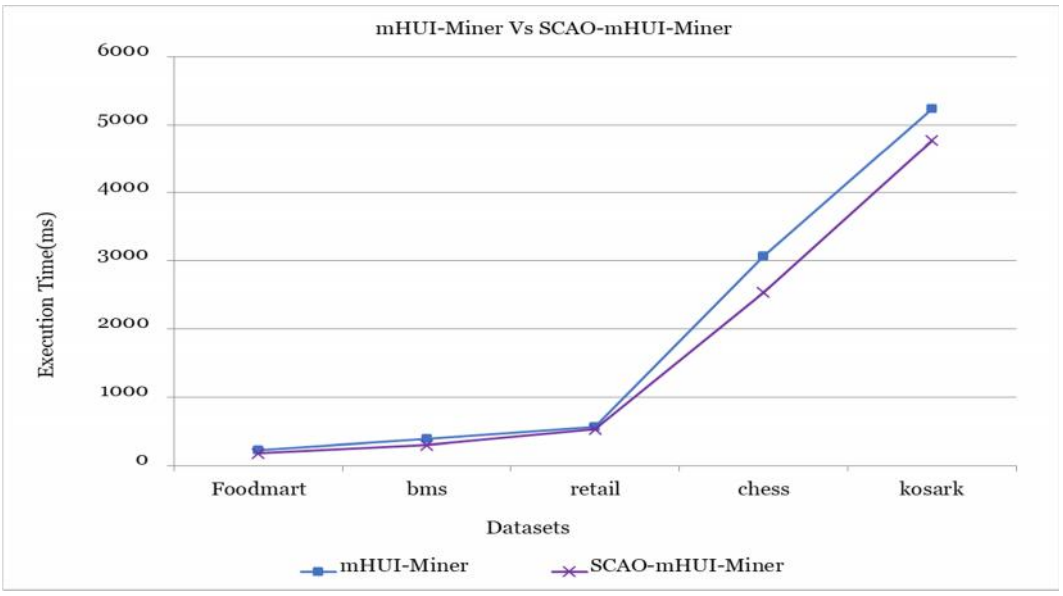


Figure 4: Execution time comparison mHUI-Miner Vs SCAO-mHUI-Miner

### 4.1.4.4 Performance Evaluation with ULB-Miner

The comparison of running time of proposed approach incorporate into ULB-Miner called SCAO-ULB-Miner and ULB-Miner as shown in Table 7 and Figure 5. The proposed technique SCAO-ULB-Miner has been found to be 10% to 24% faster than ULB-Miner.

Table 7: Execution time comparison ULB-Miner Vs SCAO-ULB-Miner

| Dataset | ULB-Miner (Execution time in ms) | SCAO-ULB-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| Foodmart | 322 | 231 | 23.62 |
| retail | 690 | 573 | 16.94 |
| bms | 829 | 680 | 17.97 |
| chess | 1170 | 1030 | 11.66 |
| kosark | 4797 | 4262 | 10.83 |

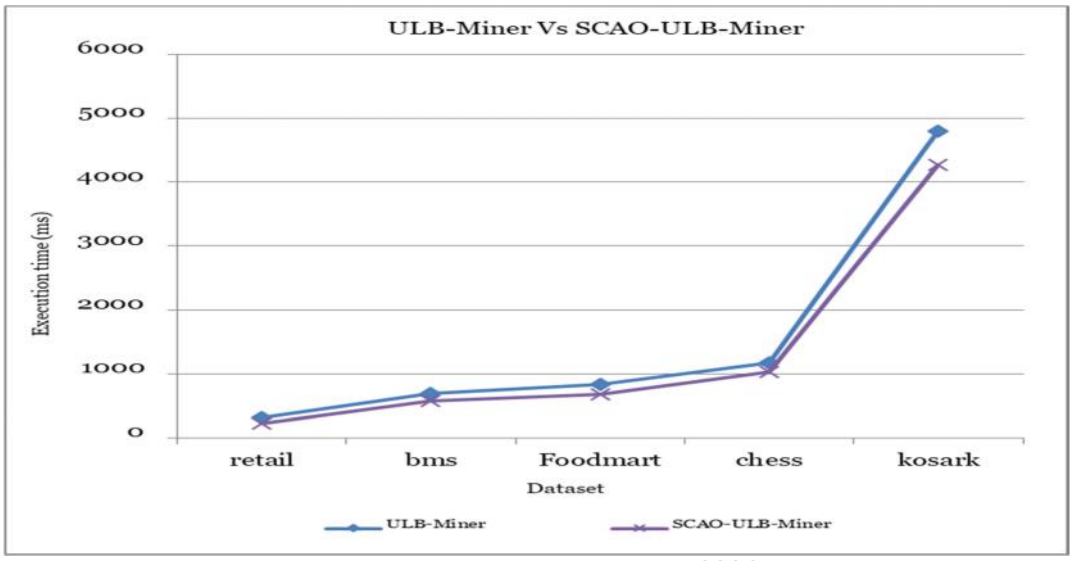


Figure 5: Execution time comparison ULB-Miner Vs SCAO-ULB-Miner

## 4.2 Predicted Utility Co-exist pruning for high utility itemset mining

Based on the items co-existence in the dataset, Predicted Utility Co-Exist Structure known as PUCS proposed to store the utility data and Predicted Utility Co-Exist Pruning known as PUCP proposed to eliminate unnecessary utility list join operations. PUCP mechanism greatly reduces the utility list join operations due to that it improves the algorithm's performance. It eliminates the low utility itemset directly without performing the join operations.  Details of proposed structure PUCS and proposed pruning method PUCP are described in the next section.

### 4.2.1 The PUCS (Predicted Utility Co-exist Structure)

A novel structure called PUCS based on the coexistence analysis of the item is shown in Figure 6. The set of triplet of the form (x, y, PU) ∈ IxIxR is known as PUCS. Predicted utility of itemset xy is represented by PU in the triplet. The PUCS was created concurrently with the creation of the initial utility list for the items during the second database scan.

$$PU = \sum_{xy \in Ti} (iutility(xy) + rutility(xy)) \tag{8}$$

Figure 7 illustrates the PUCS structure of sample dataset presented in Table 1 & 2. In this context, we define the pruning condition as "If there is no tuple (x, y, PU) in PUCS structure where PU is greater than or equal to *MinUtility*, then we consider the itemset p = {xy} and its supersets as itemsets with low utility. Consequently, there is no need to further explore these itemsets.

| | L | M | N | O |
|---|---|---|---|---|
| K | $PU_{KL}$ | $PU_{KM}$ | $PU_{KN}$ | $PU_{KO}$ |
| L | | $PU_{LM}$ | $PU_{LN}$ | $PU_{LO}$ |
| M | | | $PU_{MN}$ | $PU_{MO}$ |
| N | | | | $PU_{NO}$ |

Figure 6: PUCS Structure

| | N | O | K | L |
|---|---|---|---|---|
| M | 42 | 35 | 16 | 18 |
| N | | 33 | 33 | 23 |
| O | | | 40 | 23 |
| K | | | | 32 |

Figure 7: PUCS of the sample database

During the second database scanning, the PUCS structure was constructed along with an initial utility list of items. Within the PUCS structure, we consider the elements for item M and N, represented as the triplet <M, N, $PU_{MN}$>. Here, $PU_{MN}$ corresponds to the sum of the *iutility* and *rutility* values for the itemset {MN}. This value can be calculated during the initial database scanning process.

Additionally, we propose a unique pruning strategy called PUCP (Predicted Utility Co-exist Pruning) to minimize the number of join operations using the PUCS structure. This strategy aims to efficiently predict and remove itemsets that are unlikely to have significant utility, thereby reducing the computational burden associated with unnecessary join operations.

### 4.2.2 The PUCP (Predicted Utility co-exist Pruning)

According to lemma-1, previous algorithms like HUI-miner, mHUI-Miner and ULB-Miner trim the search space, using the addition of *iutility* and *rutility* values of an itemset. For any itemset {xy}, these algorithms construct itemset {xy}'s utility list even though it is a low utility itemset. Then decide whether itemset {xy} should be extended further based on sum of *iutility* and *rutility* values. These algorithms perform a number of costly utility list join operations for constructing low utility itemset.

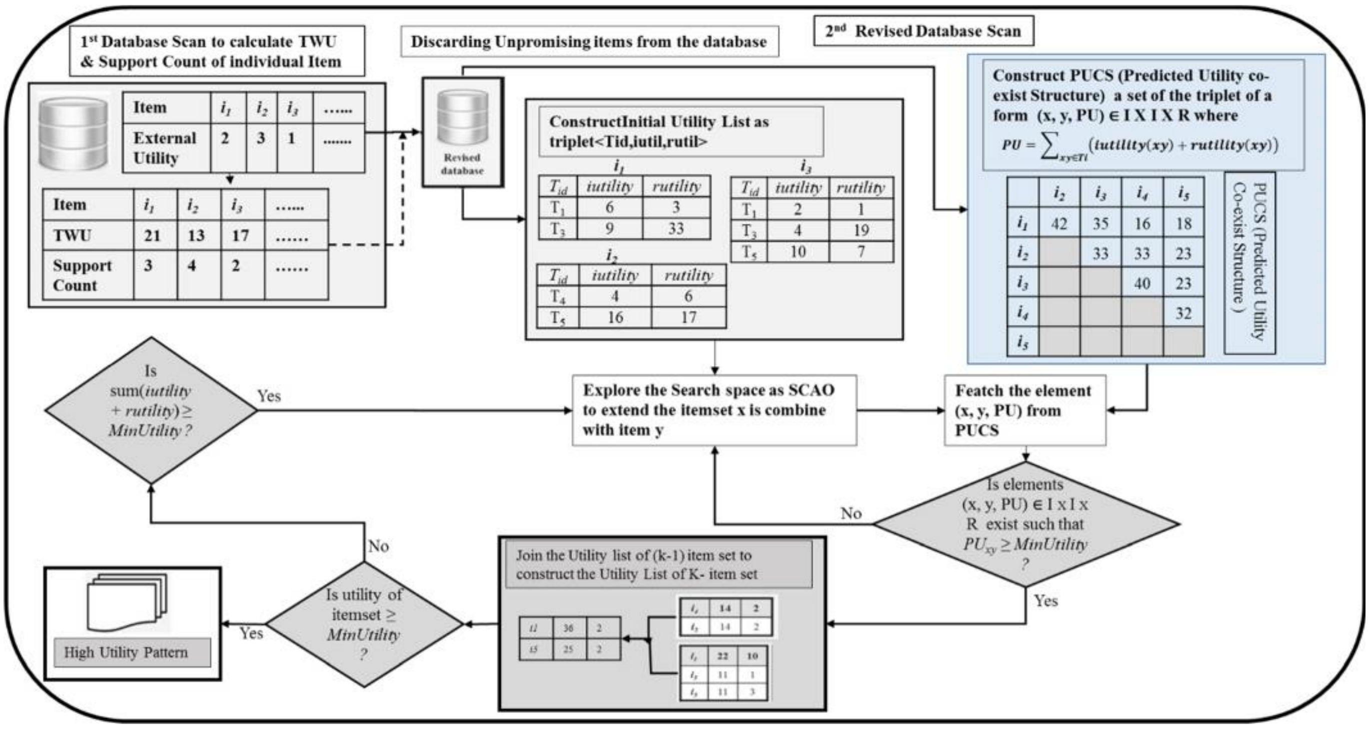
Our proposed PUCP eliminates the joining operation for the low utility itemset. For constructing the utility list of itemset {xy}, proposed algorithm called PUCP-Miner, checks the element <x, y, $PU_{xy}$> from PUCS. If an element does not exist in the PUCS where $PU_{xy} \geq$ *MinUtility,* itemset {xy} is discarded directly without constructing the utiliy list of itemset {xy}. As a result, it will minimize join operations of utility list.

Take the *MinUtility* threshold 40 as an example. To construct the itemset {MN} and its utility list, apply join operation on utility list of individual items M & N. According to PUCP, check the $PU_{MN}$ from the PUCS that is 42, so join operations has performed. While for construction of itemset {MO} the $PU_{MO}$ from the PUCS is 35, less than *MinUtility* threshold, so there is no need to perform the join operation. By removing needless join operations, this pruning method PUCP to reduce the amount of join operations significantly.

Overall procedure for discovering the high utility itemsets by proposed approach namely PUCP-Miner is represented in Figure 8.

First PUCP-Miner scans the dataset to calculate the TWU and support count of each individual item and discards the unpromising items. The items with TWU is less than the *MinUtility* threshold is considered as unpromising items. After that, it arranges the remaining items in transaction as support count ascending order (SCAO) called revised transaction. The set of revised transaction, called revised database. Revised database is scanned and it generates the initial utility list of each items and constructs the PUCS simultaneously.

The method then explore search space represented in set enumeration tree as support count ascending order to extend the itemset by combining with other item. Next, it fetches the element from the PUCS corresponding to the itemset to be extended and item to join. If PU of the element satisfies the *minUtility* requirement then the join operation is performed. Otherwise, the join operation is eliminated. The utility of the extended itemset is checked if it is higher or equal to *minUtility* then it will added in HUI list. Next, it checks the pruning condition, if the *iutility* and *rutility* of the itemset is higher than the *minUtility* threshold then the itemset is further extended. This procedure is performed recursively for exploring all itemset.


1st Database Scan to calculate TWU & Support Count of individual Item
Discarding Unpromising items from the database
2nd Revised Database Scan
Revised database
ConstructInitial Utility List as triplet<Tid,iutil,rutil>
Construct PUCS (Predicted Utility co-exist Structure) a set of the triplet of a form (x, y, PU) ∈ I X I X R where
$PU = \sum_{xy \in Ti} (iutility(xy) + rutility(xy))$
PUCS (Predicted Utility Co-exist Structure)
Is sum(iutility + rutility) ≥ MinUtility ?
Yes
Explore the Search space as SCAO to extend the itemset x is combine with item y
Featch the element (x, y, PU) from PUCS
Is elements (x, y, PU) ∈ I x I x R exist such that $PU_{xy}$ ≥ MinUtility ?
No
Yes
Join the Utility list of (k-1) item set to construct the Utility List of K- item set
Is utility of itemset ≥ MinUtility ?
No
Yes
High Utility Pattern

| Item | $i_1$ | $i_2$ | $i_3$ | ...... |
|---|---|---|---|---|
| External Utility | 2 | 3 | 1 | ....... |

| Item | $i_1$ | $i_2$ | $i_3$ | ...... |
|---|---|---|---|---|
| TWU | 21 | 13 | 17 | ...... |
| Support Count | 3 | 4 | 2 | ...... |

$i_1$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_1$ | 6 | 3 |
| $T_3$ | 9 | 33 |

$i_3$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_1$ | 2 | 1 |
| $T_3$ | 4 | 19 |
| $T_5$ | 10 | 7 |

$i_2$

| $T_{id}$ | *iutility* | *rutility* |
|---|---|---|
| $T_4$ | 4 | 6 |
| $T_5$ | 16 | 17 |

|  | $i_2$ | $i_3$ | $i_4$ | $i_5$ |
|---|---|---|---|---|
| $i_1$ | 42 | 35 | 16 | 18 |
| $i_2$ |  | 33 | 33 | 23 |
| $i_3$ |  |  | 40 | 23 |
| $i_4$ |  |  |  | 32 |
| $i_5$ |  |  |  |  |

Figure 8: Proposed Model for PUCP-Miner.

### 4.2.3 Performance Evaluation

Comprehensive experiments were conducted on a variety of real datasets, employing different *MinUtility* percentages, to evaluate performance of the proposed PUCP-Miner thoroughly. The experimental results of PUCP-Miner were compared to state-of-the-art methods including mHUI-Miner, ULB-Miner, and HUI-Miner, specifically focusing on execution time and memory requirements. Standard real-time datasets were utilized in the experiments to accurately measure the algorithm's performance. The detailed description of the datasets used in the experiments is presented in Table 6.

#### 4.2.3.1 Execution Time Analysis

Execution time of proposed approach PUCP-Miner with state of the art approaches HUI-Miner, mHUI-Miner, and ULB-Miner on different datasets are listed into Table 8, 9 and 10, respectively. Also execution time is plotted in Figures 9, 10 and 11 for performance comparison. It is observed that on ecommerce dataset, Compared to HUI-Miner, mHUI-Miner, and ULB-Miner, suggested PUCP-Miner is almost 62%, 65% and 20% faster, respectively. On the BMS dataset, PUCP-Miner takes nearly 45% less time than HUI-Miner, 46% less time than mHUI-Miner, and 42% less time than ULB-Miner. On the Foodmart dataset, PUCP-Miner is almost 67% quicker than HUI-Miner, 18% quicker than mHUI-Miner, and 18% quicker than ULB-Miner. On the retail dataset, proposed PUCP-Miner almost takes 88%, 74% and 35% less running time than HUI-Miner, mHUI-Miner and ULB-Miner respectively. Lastly, on the kosark dataset, PUCP-Miner is approximately 10% faster than HUI-Miner, 14% faster than mHUI-Miner, and 19% faster than ULB-Miner. From the execution time analysis, it is concluded that proposed PUCP-Miner outperforms on retail, BMS, eCommerce and foodmart datasets and it has a satisfactory improvement in kosark datasets.

Table 8: Execution time comparison HUI-Miner Vs PUCP-Miner

| Dataset | HUI-Miner (Execution time in ms) | Proposed PUCP-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| Foodmart | 393 | 131 | 66.67 |
| BMS | 394 | 218 | 44.67 |
| ecommerce | 1323 | 497 | 62.43 |
| kosark | 4415 | 3020 | 31.6 |
| Retail | 37857 | 4471 | 88.19 |

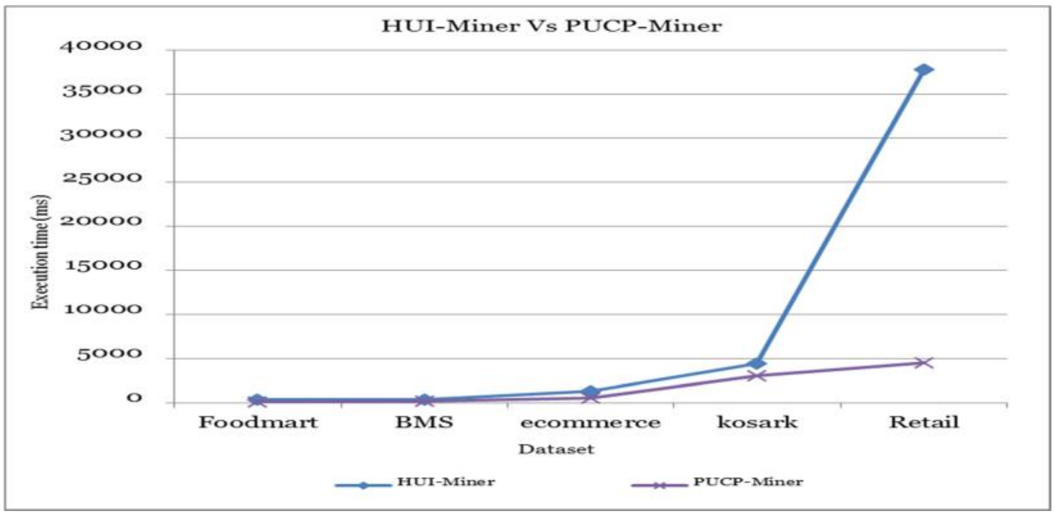


Figure 9: Execution time comparison HUI-Miner Vs PUCP-Miner.

Table 9: Execution time comparison mHUI-Miner Vs PUCP-Miner.

| Dataset | mHUI-Miner (Execution time in ms) | Proposed PUCP-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| Foodmart | 161 | 131 | 18.63 |
| BMS | 402 | 218 | 45.77 |
| ecommerce | 1434 | 497 | 65.34 |
| kosark | 4030 | 3020 | 25.06 |
| Retail | 17002 | 4471 | 73.7 |

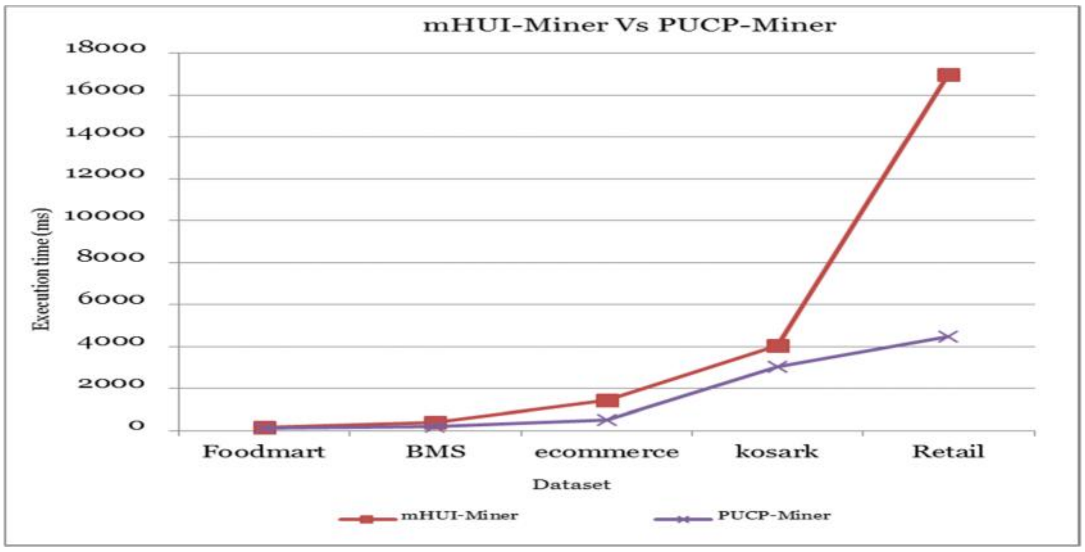


Figure 10: Execution time comparison mHUI-Miner Vs PUCP-Miner.

Table 10: Execution time comparison ULB-Miner Vs PUCP-Miner

| Dataset | ULB-Miner (Execution time in ms) | Proposed PUCP-Miner (Execution time in ms) | Improvement (%) |
|---|---|---|---|
| Foodmart | 160 | 131 | 18.13 |
| BMS | 378 | 218 | 42.33 |
| ecommerce | 624 | 497 | 20.35 |
| Kosark | 4112 | 3020 | 26.56 |
| Retail | 6897 | 4471 | 35.17 |

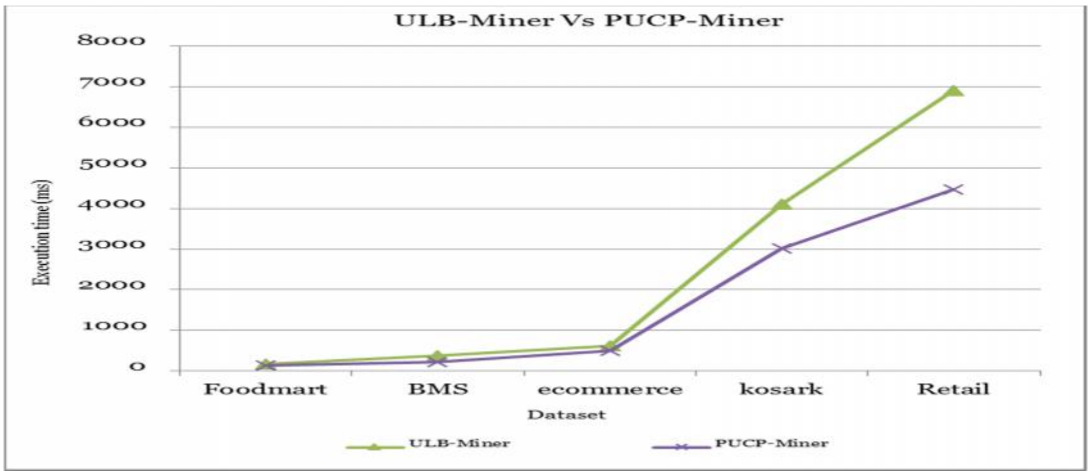


Figure 11: Execution time comparison HUI-Miner Vs PUCP-Miner

From the overall result analysis as shown in Table 11, it is observed that using only SCAO based approach improves the performance of HUI-Miner approximately 14 %, mHUI-Miner 16% and ULB-Miner 17%. While PUCP-Miner incorporating both SCAO based search space exploration and PUCP is approximately 59% faster than HUI-Miner, approximately 46% faster than mHUI-Miner and approximately 29% faster than ULB-Miner.

Table 11: Overall improvement of proposed approaches

| Algorithms | Average improvement (%) | |
|---|---|---|
| | SCAO Based Approach | PUCP-Miner (SCAO + PUCP) |
| HUI-Miner | 13.33 | 58.71 |
| mHUI-Miner | 15.74 | 45.7 |
| ULB-Miner | 17.44 | 28.50 |

### 4.2.3.2 Memory Analysis

Memory requirement of PUCP-Miner with state of the art approaches mHUI-Miner and ULB-Miner on different datasets are listed into Table 12 and 13, respectively. Also memory requirement is plotted in Figures 12 and 13 for performance comparison. The findings from the experiments indicate that PUCP-Miner utilizes significantly less memory compared to mHUIMiner and ULB-Miner on the ecommerce dataset, with reductions of approximately 46% and 8%, respectively. On the BMS dataset, PUCP-Miner consumes around 12%, and 13% less memory than mHUIMiner, and ULB-Miner, respectively. When considering the Foodmart dataset, PUCP-Miner demonstrates a memory reduction of 47% and 8% compared to both mHUI-Miner and ULB-Miner. For the retail dataset, PUCP-Miner requires 32% and 12% less memory than mHUI-Miner and ULB-Miner, respectively. Finally on the kosark dataset PUCP-Miner consumes approximately 9% and 7% less memory than mHUI-Miner and ULB-Miner.

Table 12: Memory Requirement comparison mHUI-Miner Vs PUCP-Miner

| Dataset | mHUI-Miner (memory in MB) | PUCP-Miner (memory in MB) | Improvement (%) |
|---|---|---|---|
| Foodmart | 67.3 | 42.7 | 36.55 |
| Retail | 511.78 | 349.36 | 31.74 |
| BMS | 24.04 | 20.92 | 12.98 |
| Ecommerc | 87.91 | 47.3 | 46.19 |
| Kosark | 509 | 465 | 8.64 |

Table 13: Memory Requirement comparison ULB-Miner Vs PUCP-Miner

| Dataset | ULB-Miner (memory in MB) | PUCP-Miner (Memory in MB) | Improvement (%) |
|---|---|---|---|
| Foodmart | 57.6 | 42.7 | 25.87 |
| Retail | 398.82 | 349.36 | 12.4 |
| BMS | 24.11 | 20.92 | 13.23 |
| Ecommerce | 51.17 | 47.3 | 7.56 |
| Kosark | 495.79 | 465 | 6.21 |

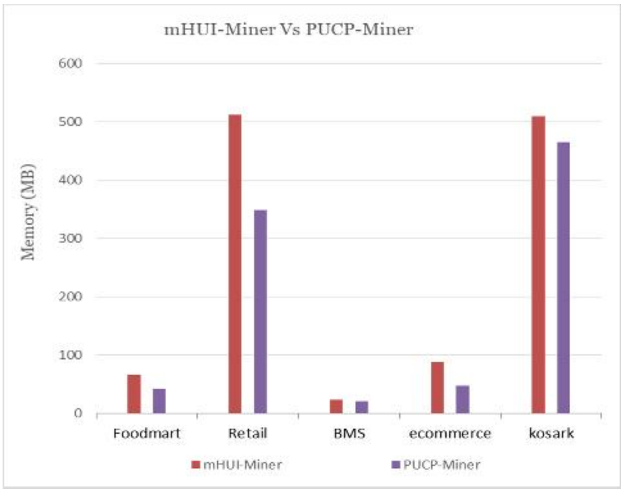


Figure 12: Memory comparison mHUI-Miner Vs PUCP-Miner

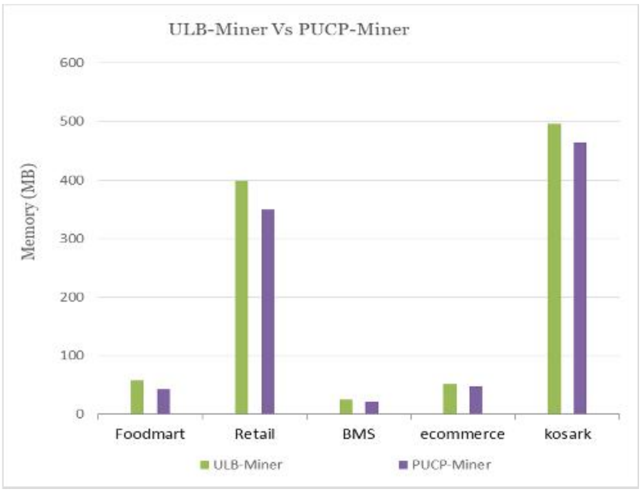


Figure 13: Memory comparison ULB-Miner Vs PUCP-Miner

## 5 Achievements with respect to objectives

The main objective of the work is to reduce the utility list join cost and join operation count to improve the performance of utility list based high utility itemset mining approach. To achieve this objective, we proposed:

(1) SCAO based search space exploration techniques to reduce the utility list join cost

(2) A novel structure called PUCS (Predicted Utility co-exist structure) and PUCP (Predicted utility co-exist pruning) to eliminate the unnecessary utility list join operation.

Hence it reduces the number of join operations. The cost of the join operation is directly related to number of comparisons required to search common transaction from both the utility lists. SCAO based approach reduces the number of comparisons. The other proposed approach namely PUCP-Miner uses the PUCS & PUCP to reduce the number of join operations. PUCP eliminates the unnecessary utility list join operations. Hence, it improves the performance of the HUIM algorithms.

## 6 Conclusion

High utility itemset mining is widely used in many business applications to discover the profitable items from the transactional dataset, identify valuable customer, and discover the significant symptoms from the medical dataset. Among the most research works carried out for HUIM, the utility list based approaches are outstanding as it does not generate the candidate itemset. However the performance of the utility list based approaches are limited due to costly utility list join operations. These approaches also perform unnecessary utility list join operations for the low utility itemsets. The cost of the join operation is directly related to the number of comparisons required to search the common transactions between utiliy lists. SCAO based search space exploration techniques greatly reduces such comparisons. Hence, it improves the performance of the HUIM approach. PUCP (Predicted Utility Co-exist Pruning) uses the PUCS (Predicted Utility Co-exist Structure) to eliminate the unnecessary utility list join operations for the low utility itemsets. PUCP decides in advance whether it is necessary to perform join operations or discard the itemset, it reduces the number of join operations. The experimental results shows that SCAO based search space exploration techniques improve the performance of the HUIM approach from 13 to 18 percent. While the combination of both SCAO based search space exploration techniques and PUCP called PUCP-Miner greatly improve the performance of the HUIM approach from 28 to 59 percent.

## 7 Research publications

1. Patel Suresh B, Sanjay M. Shah, and Mahendra N. Patel. "An Efficient High Utility Itemset Mining Approach using Predicted Utility Co-exist Pruning." International Journal of Intelligent Systems and Applications in Engineering 10, no. 4 (2022): 224-230. **(SCOPUS Approved, ISSN: 2147-6799)**

2. Patel Suresh B, Sanjay M. Shah, and Mahendra N. Patel. "An Efficient Search Space Exploration Technique for High Utility Itemset Mining." Procedia Computer Science 218 (2023): 937-948. (**SCOPUS Approved, ISSN: 1877-0509**)

3. Patel, Mahendra Narottamdas, Sanjay M. Shah, and Suresh B. Patel. "An Adjacency matrix-based Multiple Fuzzy Frequent Itemsets mining (AMFFI) technique." International Journal of Intelligent Systems and Applications in Engineering 10, no. 1 (2022): 69-74. **(SCOPUS Approved, ISSN: 2147-6799)**

4. Patel, Mahendra N., S. M. Shah, and Suresh B. Patel. "An Efficient (MFFPA-2) Multiple Fuzzy Frequent Patterns Mining with Adjacency Matrix and Type-2 Member Function." In International Conference on Advances in Computing and Data Sciences, pp. 502-515. Cham: Springer Nature Switzerland, 2023.

## 8 References

[1] D.-N. Le Ashour, Amira S., Nilanjan Dey, “Biological data mining: Techniques and applications,” Min. Multimed. Doc., vol. 1, no. 4, pp. 161–172, 2017.

[2] W. Z. Cheng and X. Li Xia, “A fast algorithm for mining association rules,” Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS, pp. 513–516, 2014.

[3] H. Yao and H. J. Hamilton, “Mining itemset utilities from transaction databases,” Data Knowl. Eng., vol. 59, no. 3 SPEC. ISS., pp. 603–626, 2006.

[4] J. Hu and A. Mojsilovic, “High-utility pattern mining: A method for discovery of high-utility item sets,” Pattern Recognit., vol. 40, no. 11, pp. 3317–3324, 2007.

[5] Y. Liu, W. K. Liao, and A. Choudhary, “A two-phase algorithm for fast discovery of high utility

itemsets," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3518 LNAI, pp. 689–695, 2005.

[6] X. Liu, P. Du, and X. Qiao, "Study on effect of master DOF on errors of substructure static condensation modal analysis," Zhongguo Jixie Gongcheng/China Mech. Eng., vol. 22, no. 3, pp. 1–12, 2011.

[7] H. Ryang, U. Yun, and K. H. Ryu, "Fast algorithm for high utility pattern mining with the sum of item quantities," Intell. Data Anal., vol. 20, no. 2, pp. 395–415, 2016.

[8] V. Tseng, C. Wu, B. Shie, and P. Yu, "UP-Growth: an efficient algorithm for high utility itemset mining," in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, 2010, pp. 253–262.

[9] Y. Shen, "Objective-Oriented Utility-Based Association Mining," in In 2002 IEEE International Conference on Data Mining, 2002. Proceedings, 2002, pp. 426–433.

[10] W. Jentner and D. A. Keim, Efficient Algorithms for High Utility Itemset Mining Without Candidate Generation, vol. 51. Springer International Publishing, 2019.

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

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

[13] A. Y. Peng, Y. S. K. B, and P. Riddle, "mHUIMiner : A Fast High Utility Itemset Mining Algorithm for Sparse Datasets," pp. 196–207, 2017.

[14] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," ACM Int. Conf. Proceeding Ser., pp. 55–64, 2012.

[15] H. Yao, H. J. Hamilton, and C. J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases," Proc. 2004 SIAM Int. Conf. Data Min., vol. Society fo, pp. 482–486, 2004.

[16] Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," Data Knowl. Eng., vol. 64, no. 1, pp. 198–217, 2008.

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

[18] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," Springer, Cham. pp. 83–92, 2014.

[19] Ahmed, Chowdhury Farhan, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee. "An efficient candidate pruning technique for high utility pattern mining." In Advances in Knowledge Discovery and Data Mining: 13th Pacific-Asia Conference, PAKDD 2009 Bangkok, Thailand, April 27-30, 2009 Proceedings 13, pp. 749-756. Springer Berlin Heidelberg, 2009.

[20] Fournier-Viger, P., Lin, J. C. W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., & Lam, H. T. (2016, September). The SPMF open-source data mining library version 2. In Joint European conference on machine learning and knowledge discovery in databases (pp. 36-40). Springer, Cham.

.