Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 218 (2023) 937–948

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Machine Learning and Data Engineering

An Efficient Search Space Exploration Technique for High Utility Itemset Mining

Suresh B Patel [1], Sanjay M Shah[2], Mahendra N Patel[3]

[1] *PhD Scholar, Gujarat Technological University, Ahmedabad, India*
[2]*Computer Engineering, L.D.C.E, Ahmedabad, India*
[3]*Information Technology Department, GEC Gandhinagar, Gujarat-India-380028.*

Abstract

Data mining techniques discover the potentially helpful pattern concealed in a vast database for the decision support system in various real-life applications. One such technique is association rule mining. It finds the pattern from the transaction database to understand customer behavior. Frequent itemset mining (FIM) identifies a set of frequently purchased items together. One key drawback of FIM is that it ignores the item's importance. An item's importance plays a pivotal role in a real-world application. Therefore, it is necessary to discover the essential itemset from the transaction database that generate the high profit called the HUIM (High-Utility Itemset Mining) problem. From a transaction database, a variety of strategies can be used to find high utility itemset. The HUIM techniques that use the Utility list are recent and have better performance in terms of memory utilization and running time. The key limitation of these algorithms is performing costly utility list join operations. This paper proposes an efficient search space exploration technique to lessen the number of comparisons performed for utility list join operations. Hence, it reduces the cost of utility list join operations. In addition, the execution time of the proposed method is evaluated. Further, the same is compared with appropriate existing state-of-the-art methods. Finally, extensive experiments carried out on publicly available benchmark datasets show that the proposed support count ascending order (SCAO) based approach performs much better as oblivious to the state-of-the-art methods.

Keywords: Mining algorithm, High utility itemset mining, pattern, utility-list, Transaction weighted utity;

1. Introduction

Today, data is treated as an asset. A massive amount of data is available in each field of real life. Using the potential information hidden inside that open data, every business organization, government institutes, and servicing organizations design the decision support system. Data mining discovers this helpful information from many available data. Frequent itemset mining is the fundamental approach to discovering the vital pattern from the transaction database, which is essential to many applications. FIM finds frequent itemsets, i.e., items purchased together frequently from the transaction database [1]. The main limitation of the FIM is that it considers only the item's presence/absent. It ignores the item's profit/Utility [1]. A customer may purchase different quantities of items in the real world, and items have different profits, importance, or utility. FIM maybe discovered frequent but less important (low profitable) and miss rare but more important item-set [1]. The HUIM addresses this problem of FIM [2, 3, 4]. HUIM discovers high profit item-sets. It considers item importance (profit/utility/weight) and quantity in the transaction database [2, 3, 4]. It determines the application's essential items based on the frequency and utility.

HUIM problem is derived from the FIM, however, due to the non-following of downward closure property, the HUIM poses higher challenges than FIM. FIM makes use of the apriori characteristic, to reduce the search space [1]. Unlike the FIM utility measures cannot be directly applied to trim the search space. The subsets of HUI may be a

1877-0509
10.1016/j.procs.2023.01.074

low utility itemset and vice-versa, making the reduction of search space difficult [2, 3, 4, 5, 6]. For a transaction database, numerous methods are devised to discover the high utility itemsets using various data structures and pruning strategies. Amongst them, the most recent and efficient approach is the utility list-based approach [12]. This utility list-based algorithm's performance in terms of running time degrades due to numerous costly utility list join operations. The cost of joint operations is determined by the number of comparisons needed to find the common Transaction in both utility lists. This paper proposed efficient search space exploration techniques in a less costly way. It reduces the number of comparisons needed for utility list join operation. Finally, extensive experiments carried out on publicly available benchmark datasets. Experimental results show that the proposed support count ascending order (SCAO) based approach performs much better in execution time as oblivious to the state-of-the-art approaches**.** The paper has been organized into 6 sections. Section 2 includes the problem definition and background. Section 3 describes prior art. The proposed method is explained in section 4. In section 5 performance study and experimental results are presented. At last, Section 6 presents the conclusion and scope for future work.

2. Problem Definition and Background

2.1. Preliminaries

Consider an item-set I, which contains $i_1,i_2,i_3 \ldots i_n$ an individual item. A transactional database-DB consists of utility information and transactions that define the quantities of an item. $T = (T_1, T_2, T_3 \ldots T_m)$ is a transaction set where the unique Transaction ID T_{id} of all entries. Each Transaction $T_i \subseteq I$ and counts $q(i_k)$ (where $1 \leq k \leq n$) showing the quantities of item i_k in T_i referred as internal utility. The external utility of item is shown in Table 2.

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. Utility Table of sample database

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

Definition 1: Transaction Database.

For itemset, $I = \{i_1, i_2, i_3, \ldots i_n\}$, where n is the total number of items. A Transaction database consists of Transaction set $T = (T_1,T_2,T_3 \ldots T_m)$ with total m transactions, each Transaction T_j is consists of a set of items ($i_k \in T_j$) associated with quantity and a profit value p (i_k) of item ik, ($i_k \in I$) shown in Utility Table is called an external utility. Table 3 and 4 show the transaction database DB of five transactions and item-set I. Transaction T3 consists of items k, l, o, and q with item quantities 3,4,2, and 1 correspondingly. While 5,1,2,2 shows the external utility of such items.

Definition 2: Item Utility in a Transaction.

The item utility in Transaction T_j for item i_k is defined as $u(i_k, T_j) = q(i_k, Tj) \times p(i_k)$ where $q(i_k, T_j)$ is the quantity of item i_k in Transaction T_j.

Example. The utility of item n in T_5 is u (m, T_5) is 9

Definition 3: itemset utility in a Transaction.

For a transaction, T_j and itemset X ($X \subseteq I$, $X \subseteq T_j$), the itemset X's utility in Transaction T_j is the total of each item's utility of X in Transaction T_j and it is denoted as $u (X, T_j) = \Sigma i_k \in x\ u(i_k, T_j)$

Example. In transaction T5, the utility of an itemset X {m, n} is $u ((m, n), T_5) = u(m, T_5) + u(n, T_5) = 9 + 16 = 25$.

Definition 4: itemset utility

For itemset X ($X \subseteq I$, $X \subseteq T_j$) and database DB, the itemset's utility is the total of itemset's utility in each Transaction containing X. It is written as $u(X) = \Sigma_{Tj \in g(x)}\ u(X, T_j)$, where X is the part of the Transaction set g(x).

Example. The utility of the itemset {kl} in database is u(k,l) = u((k,l),T3) + u((k,l),T4) + u((k,l),T5) = 32

Definition 5: High Utility Itemset Mining Problem.

HUIM problem is to find all the itemsets whose utility is greater than the user's minimal utility threshold minUtil. If the utility u(X) of an item-set X is more than or equal to minUtil, it is referred to as a high utility item-set.

High Utility item-set = $X \subseteq I$ where $u(X) \geq$ MinUtil

Example. With MinUtil=15 and item-set X = (k,l), u(k,l) is more than MinUtil, indicating that X is a high utility itemset.

2.2. HUIM's Challenges

Because of the lacking of monotonic and downward closure properties, the HUIM problem is widely accepted as more complex than the FIM problem in terms of running time and storage. The downward-closure-property in FIM asserts that an itemset's frequency is anti-monotonic [2]. By using the same property, the search space can be trimmed efficiently. [1]. But item's utility in HUIM is neither monotonous nor anti-monotonous. Thus the utility of itemset's subset or superset is not predictable as lesser or higher than the itemset's utility [7, 14, 15, 16]. As a result, the pruning methods employed in FIM cannot be used to in High Utility itemset mining to trim the search space. i.e, Consider the database in table 3 and the utility values in table 4, respectively, with MinUtil= 40. According to definition 5, the item set x = k,m,n,o has a utility value of 40, making it a high utility item set. While item set y {k,m,n} 's utility is 30, it is not a high utility item set even $y \subseteq x$. Therefore, reducing the search space in HUIM is a more challenging than in FIM.

3. Prior Art

Discovering the High utility itemsets is harder than FIM as there is no direct measure same as the support count in FIM to trim the search space. HUIM finds profitable items from the sales database, identifies valuable clients, and selects the significant symptoms from the medical dataset that can be used to design a reliable decision-making system. So It's a highly demanded research point in the world of data mining, and it's attracting a lot of attention. Because HUIM lacks monotonic and downward closure features, most past research has concentrated on shrinking the search space, which is difficult. Based on different mining ideologies, data structure, and pruning strategies, HUIM algorithms are mainly classified into three classes (1) candidate generation and test based [2, 3, 4, 5]. (2) Tree based[6,7 , 8, 9, 10, 11] (3) Utility List based algorithms. Many of HUIM algorithms from the first two categories suffer from a massive number of candidate generations, and overestimating the pruning measure leads to more time-consuming and memory [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Utility list-based algorithms are recent and mine HUI without candidate generation.

Definition6: Transaction Utility (TU)

The total of all item's utility in T_j is called Transaction Utility. it is defined as $TU(Tj)= \Sigma_{ik \in Tj} u(i_k, T_j)$. For example $TU(T_1)$ is the sum of utility of items l,m, and o in transaction T_1 that is 9.

Definition7: An itemset's Transaction Weighted Utility (TWU)

The total of Transaction's utility $TU(T_c)$ of all transactions(T_c) containing itemset P is called Transaction-Weighted-Utility(TWU).

$$TWU(P) = \sum_{P \in Tc} TU(T_c).$$

The whole profit from all transactions that contain the itemset P is represented by TWU.
Example 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

Definition 8: Low Transaction Weighted Utility itemset

If the TWU(X) is less than the user-specified minimum utility threshold *minUtil*, then itemset X is the Low Transaction Weighted utility itemset.

3.1. Utility List-Based Approaches

HUI-Miner, the first single-phase algorithm for HUIM without candidate creation, was proposed by Liu and Qu in 2012. It solves the enormous amount of candidate generation problems of previous apriori-based and tree-based

approaches. HUI-Miner stores the itemset's utility information in the novel list structure. It is named as Utility List. It is recursively discover the high utility k-itemsets by performing the join operations on (k-1) itemsets[12]. Although HUI-Miner is faster than earlier methods, it still executes time-consuming utility list join operations. To decrease the join count in 2014, Viger, Wu, Zida, and Tseng proposed FHM using EUCP. It deals with costly utility list join operations in HUI Miner. Based on item co-occurrence They presented Novel pruning method called Estimated Utility Co-occurrence Pruning (EUCP) to minimize utility list join operations count utilising the EUCS[13]. In 2017, Peng, Koh, and Riddle proposed a tree structure to avoid generating utility lists of the item sets that do not exist in the database. Due to that, these two algorithms are somewhat inefficient. Modified HUIMiner that incorporate tree structure IHUP into HUIMiner. As per IHUP Tree Structure-property, the Transaction stored as a path of the tree. This implies that all the information about the item's togetherness is stored in the tree[14]. In 2017, Duong, Viger, ramampiaro, norvag, and dam proposed efficient HUIM using utility list buffer to store the item's utility information. The ULB structure reuses the memory of the item set that will not be further expanded[15]. In 2019, Qu, Liu, and viger proposed an improved performance of HUI-Miner by proposing the new structure Utility-List*. Based on the observation, the performance of HUI-Miner, FHM, mHUIMiner and ULB-Miner degrade due to ineffective Tids comparisons for joining the utility lists. The HUI-Miner* removes these ineffective comparisons by Utility-list* structure[16].

4. Proposed Method

The proposed SCAO-based HUIM process model is as shown in Figure 1. The proposed approach work in three phase.

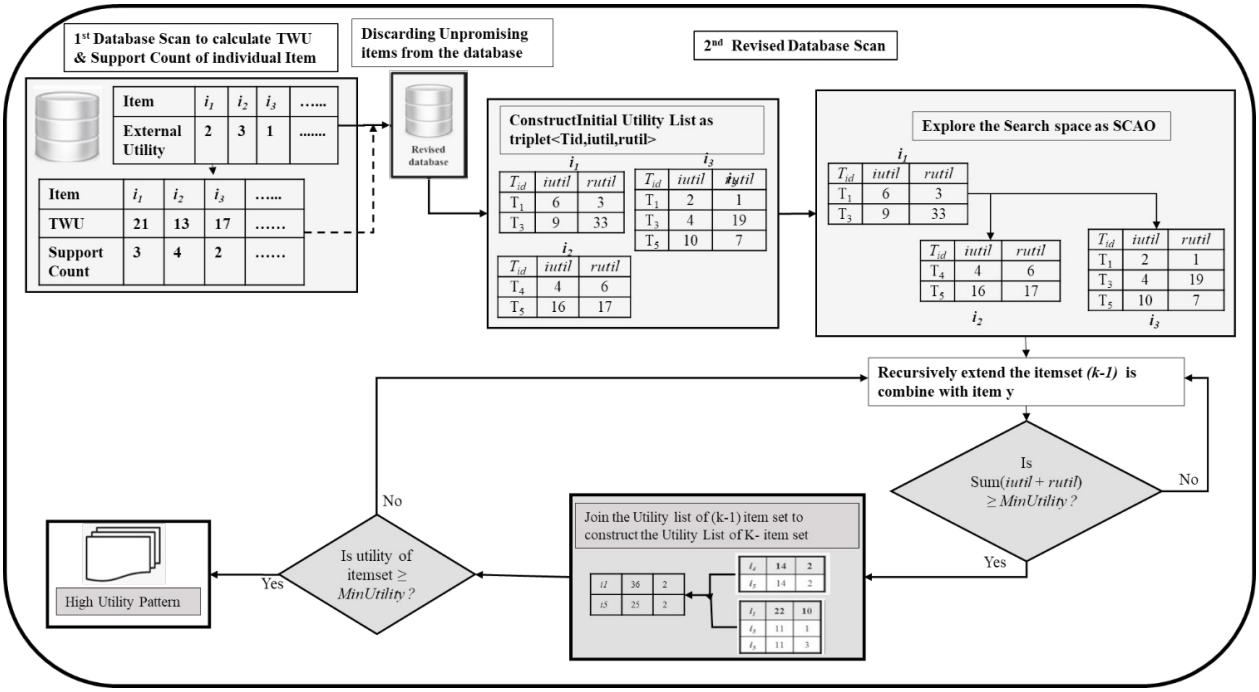

Fig.1 SCAO based HUI Mining process

Phase 1: construction of a revised database

Scan the dataset to compute TWU and support count of each item. They were then discarding the unpromising items which have lesser TWU than MinUtil. Scan the database again to rearrange all transactions in support count ascending order called revised Transaction. The set of revised Transactions is called a revised database. Take the data in table 2 & 4, set the MinUtil as 40. First, go through the database and determine each item's TWU. Discard the unpromising items which have less TWU than MinUtil threshold. Here items *p* and *q* are discarded, as they are

unpromising items. Rearrange the items in the Transaction as Support count ascending order e.g m-n-o-k-l is called a revised transaction.

Phase 2: Construction of initial utility list

Scan the database again to construct the initial utility list of each promising item as per algorithm 1. A utility list of items is the set of triplets associated with the Transaction the item contains. The triplet <TRid, iutil, rutil> where TRid is the Item's transaction ID. Iutil is item's utility value in a revised Transaction. rutil is an item's remaining utility value[12, 13, 14, 15, 16].

Definition 9: All the items in Transaction T that come after itemset X where $X \subseteq T$ is denoted as T|X. consider T5 | mn = {okl} and T3|o = {kl}

Definition 10: Remaining utility

The total utility of all the items that come after itemset X in transaction T, is represented as ru(X, T).

$ru(X,T) = \sum_{\mathbf{i} \in (\mathbf{T}|\mathbf{X})} \mathbf{u}(\mathbf{i}, \mathbf{T})$ where $X \subseteq T$

.i.e., constructing the item o's utility list in transaction T_3. iutil value of *o* in T_3 is 4 and remaining utility rutil value of *o* in T_3 = ru(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 in table 5.

Phase 3: Search space exploration & mining process

A set enumeration tree, as in fig 2, can be used to represent the search space. After creating the initial utility list, the proposed technique explore the search space as Support count ascending order (SCAO). It recursively extends the (k-1) itemset by combining with the successor item. It performs the pruning based on the sum of iutil and rutil to decide whether the itemset further extended. The details of the pruning mechanism as in section 4.2. Construct the utility lists of k-itemset as algorithm-2 by joining the utility lists of (k-1) itemset. The details of the utility list join operation as in section 4.3. Simultaneously as per algorithm 3, it discovered the high utility itemsets.

Table 3. Utility List of 1-Itemset

{m}

Tid	Iutil	rutil
T_1	6	3
T_5	9	33

{n}

Tid	Iutil	rutil
T_4	4	6
T_5	16	17

{o}

Tid	Iutil	rutil
T_1	2	1
T_3	4	19
T_5	10	7

{k}

Tid	Iutil	rutil
T_3	15	4
T_4	5	1
T_5	5	2

{l}

Tid	Iutil	rutil
T_1	1	0
T_3	4	0
T_4	1	0
T_5	2	0

Table 4. Utility List of 2-Itemset

{mn}

Tid	Iutil	rutil
T_5	25	17

{mo}

Tid	Iutil	rutil
T_1	8	1
T_5	19	7

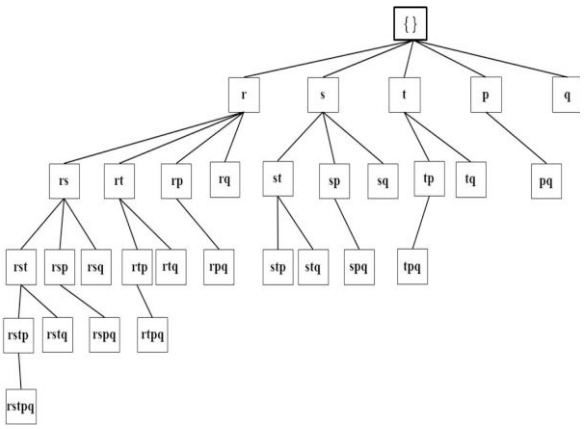

Fig.2.Set Enumeration Tree

4.1. Construction of Utility list of 2-itemset and k-itemset

Consider the 2-itemset x= {mn}, itemset x's utility list constructed by performing the join operations on i-itemset m's utility list and n's utility list. .For the join operation, it searches the common Transaction from both the utility list and adds the element<Tid, iutil, rutil> into the utility list of {rs}. Where Tid is the common transaction ID, iutil is the summation of iutil of {m} and iutil of {n}, rutil is the rutil value of {n} as itemset {n} comes after itemset{m} as per SCAO (Support count ascending order) for example construction of {mn}'s utility list, it join the m's utility list and n's utility list. There are one transaction id T5 is common in both the utility list, so add the element <T5, 25, 17> in {mn}. Similarly, add all common transactions in {mn}. The utility list of 2-itemset as in Table: 4. In the same way, k-itemset's utility list can be constructed by joining the two (k-1) itemset's utility list.

Algorithm 1:- Mining Algorithm

Input: -LUL - List of UtilityList of 1-itemset
Prev.UL- previous item's utiitylist initially it is empty,
MU – User specific Minimum utility threshold
Output: -high-Utility itemsets

```
1.  For each element K in LUL do
2.      If TOTAL of K's iutil ≥ MinUtil then
3.          Add K & its previous itemsets in resultset
4.      endif
5.      If TOTAL of all K's iutil & rutils ≥ MU then
6.      Create empty extUL;
7.          For each element L follow K in LUL do
8.              extUL=extUL+ join(Prev.UL,K,L)
9.          endfor
10.         Mining(K,extUL,MU)
11.     endif
12. endfor
```

Algorithm 2:- Join Algorithm

Input: - UL_{Prev}- itemset Prev 's utility list.
UL_K– itemset K's utility list,
UL_L – itemset L's utility list
Output: - UL_{KL}- itemset KL's utility list.

```
1.  Initialize ULKL is NULL
2.  foreach component Ek ∈ ULK
3.      if ∃ component EL ∈ULL&& Ek.Tid == EL.Tid then
4.          if ULPrev ≠ empty then
5.              Search component EPrev∈ ULPrev have same Tid
6.              Create new component Where EkL = <Ek.Tid,
    Ek.iutil+EL.iutil – Eprev.iutil,EL.rutil>
7.          else
8.              EkL= <Ek.Tid,Ek.iutil+EL.iutil – Eprev.iutil,EL.rutil>
9.          endif
10.         Insert component EkLto ULKL
11.     endif
12. Endfor

    return ULKL
```

4.2. Prunning Mechanism

In-depth search space exploration identified all high utility itemset, but it consumes more time because many items are available in many datasets. Therefore, it is necessary to trim the
itemset that does not contribute to the high utility. Use the itemset utility *iutil* and remaining itemset utility *rutil* values from the itemset's utility list to narrow the search field. The total of iutil 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 MinUtil criterion. Itemset can be further extended if its sum of *iutil* and *rutil* value is more or equal to MinUtil threshold. Otherwise, it can be discarded as per lemma 1[16].

Lemma 1:- If the sum of all iutil and rutil values from X's utility list UL(X) is no less than the minUtil, 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 utilities and remaining itemset utilities (iutil & rutil respectively) is 42, which is larger than the *minUtil* Threshold 40. So it can be further extended. While the total of iutil and rutil values of {mo} is 35, so it can prune the {mo}.

4.3. Analysis of Proposed Method Vs. state of the art methods

SCAO-based algorithms require fewer comparisons for utility list join operations. Time complexity analysis shows that proposed SCAO-based algorithms need more occasional comparisons to join utility lists. Assume UL_a, UL_b, UL_c are the utility list of 1-itemset *a,b* and *c*. p, q and r is the numbers of entries in utility lists (support count)UL_a,UL_b, UL_c, respectively. Where $p \geq q \geq r$. For Construct, the itemset *ab*'s utility list performs the join

operation of UL_a, 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 ULab and ULac, 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* $UL_{cba,}$ so 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 ULcb and ULca, 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 t*otal 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 asUL_{ca}, UL_{cb}, $UL_{cab.}$ 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 operation.

5. Performance Study

The researchers incorporated the proposed approach to explore the search space as support count ascending order (SCAO) in HUI-Miner, mHUI-Miner, and ULB-Miner algorithms. Perform extensive experiments on various real datasets with different MinUtility percentages.

5.1. Experimental Environment

All the experimental algorithms have been implemented in JAVA. All the tests were performed on system having 8GB RAM and Intel core i5, 3.4 GHz processor running windows 10 pro. Standard real-time datasets [17] were used in the trials to assess the algorithm's performance on different minUtil. Table 7 lists the dataset's properties and a full description. The number of items, transactions, and Transaction length varied amongst the datasets.

Table 5. Characteristics of Dataset

Sr.No	Dataset Name	#Transactions	#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

5.2. Experimental result analysis

5.1.1 Performance Evaluation with HUI-Miner

Compared Algorithms are run on various datasets with various utility thresholds decreasingly until it takes too long or memory is full and the execution time is recorded. The comparison of the running time needed for the proposed approach explores the search space as SCAO called SCAO_HUI-Miner and HUI-Miner as shown in figure 3. The proposed technique SCAO HUI-Miner is 4 to 16 percent better perform than HUI-Miner in terms of execution time.

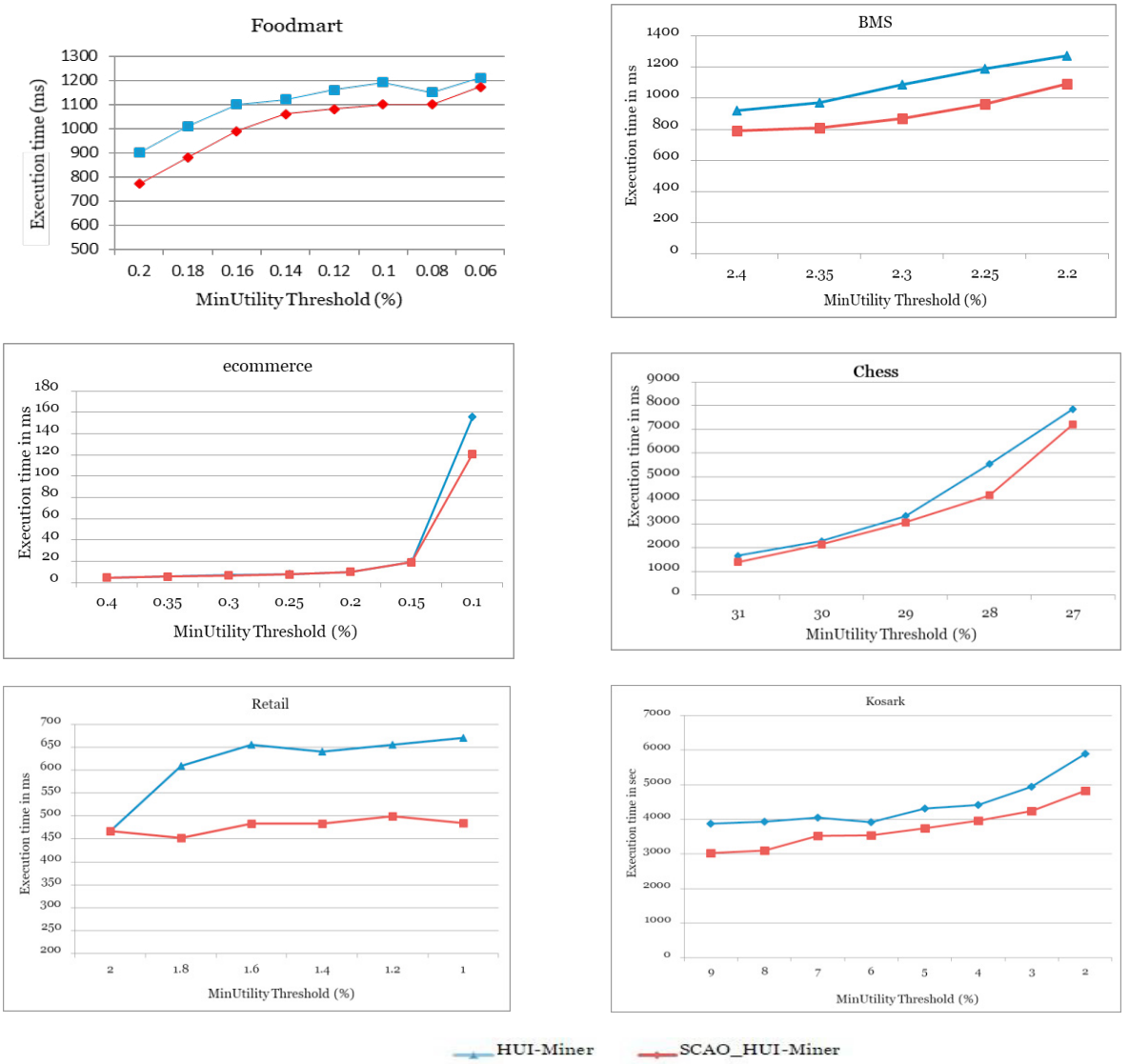

Fig.3. Running time of HUI-Miner Vs SCAO_HUI-Miner

5.1.2 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 mHUI-Miner as shown in figure 4. Proposed technique SCAO mHUIMiner has been found to be 6 to 18% faster than mHUI-Miner in terms of execution time.

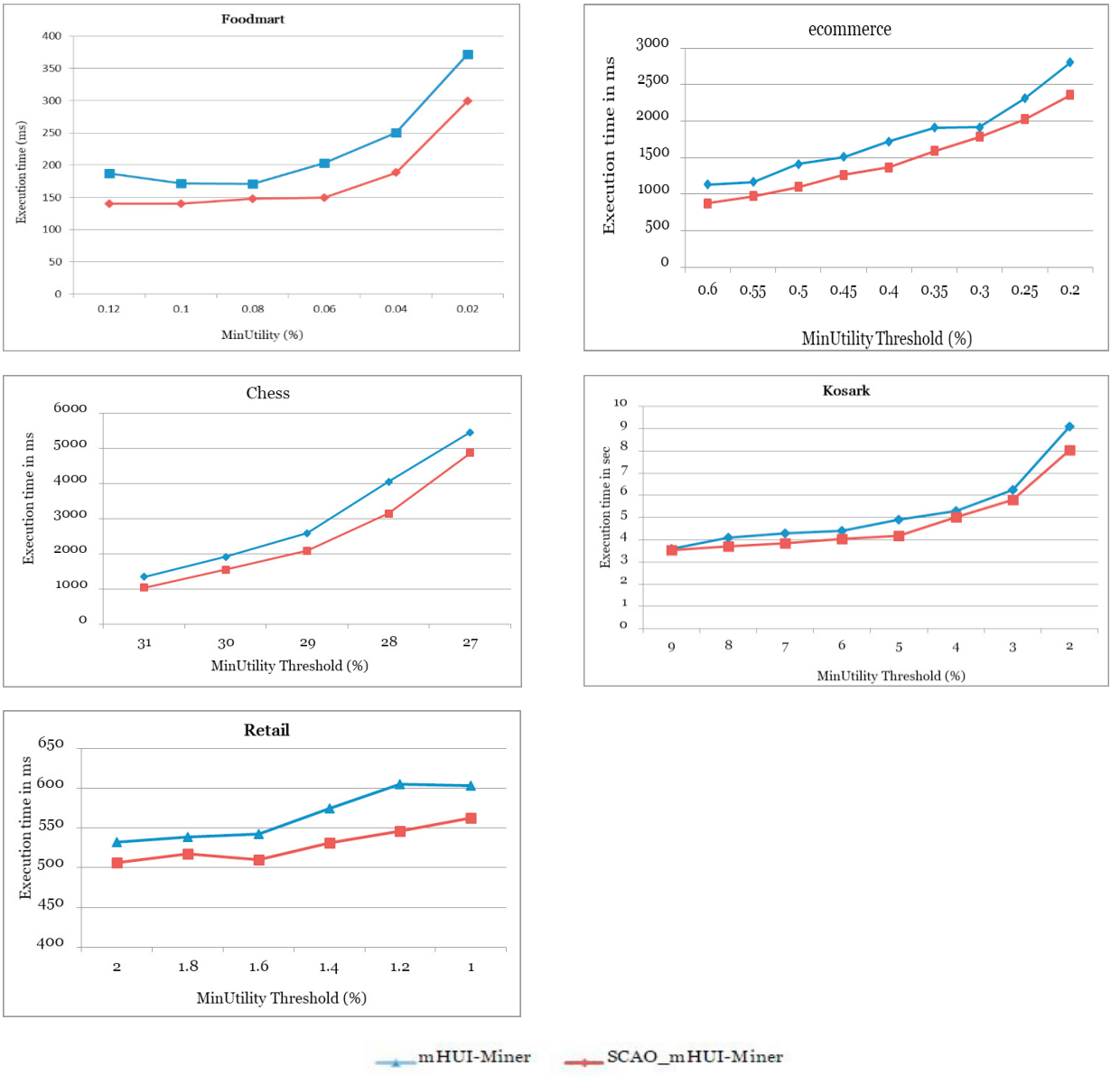

Fig.4. Running time of mHUI-Miner Vs SCAO_mHUI-Miner

5.1.3 Performance Evaluation with ULB-Miner

The comparison of running time employs ULB-Miner called SCAO_ULB-Miner and ULB-Miner as shown in figure 5. The proposed technique SCAO_ULB-Miner has been found to be 10 to 24% faster than ULB-Miner.

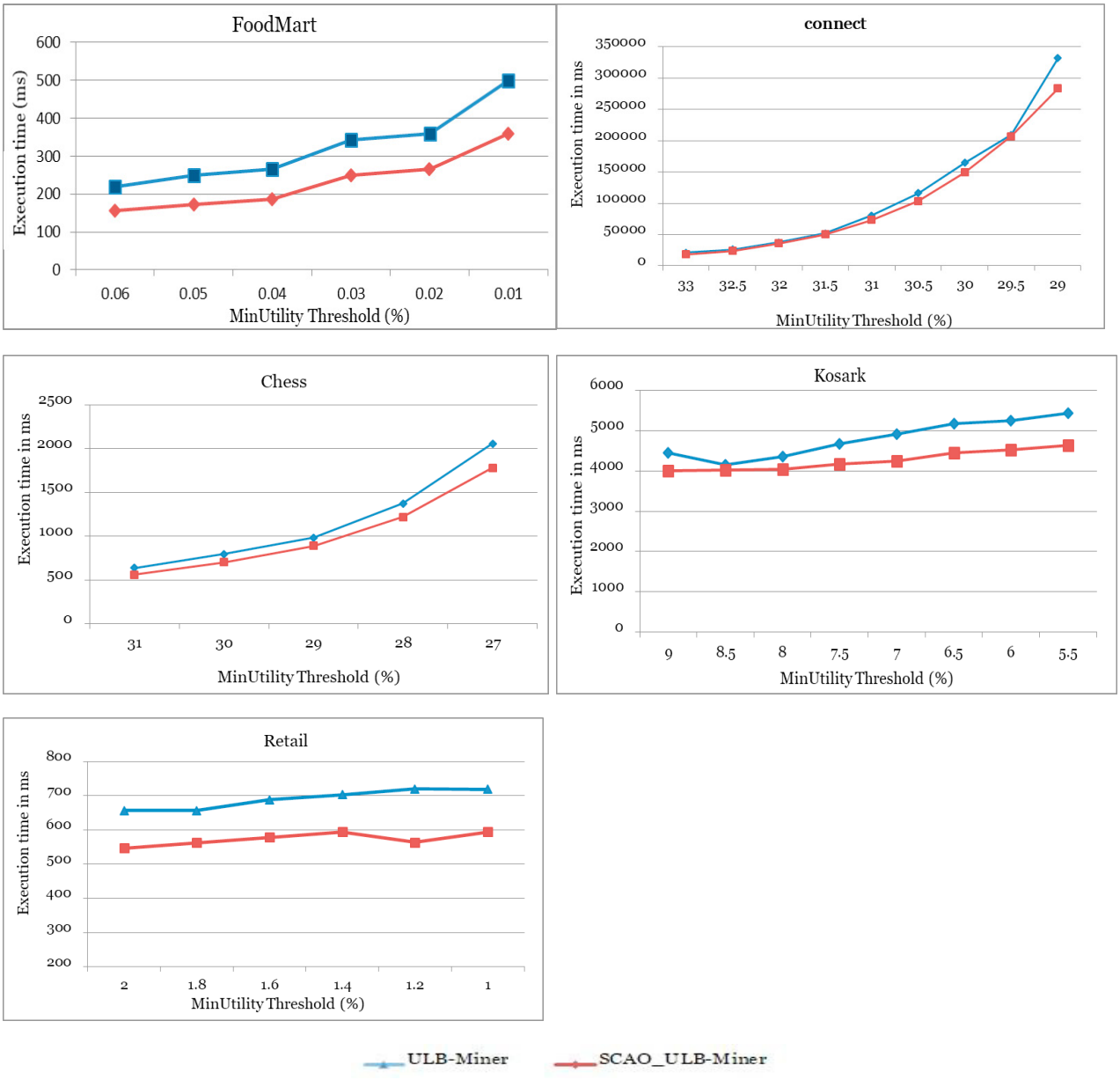

Fig.5. Running time of ULB-Miner Vs SCAO_ULB-Miner

5.1.4 Performance Evaluation with FHM

The comparison of running time employs FHM called SCAO_FHM and FHM as shown in figure 6. The proposed technique SCAO_FHM has been found to be 15 to 26% faster than FHM.

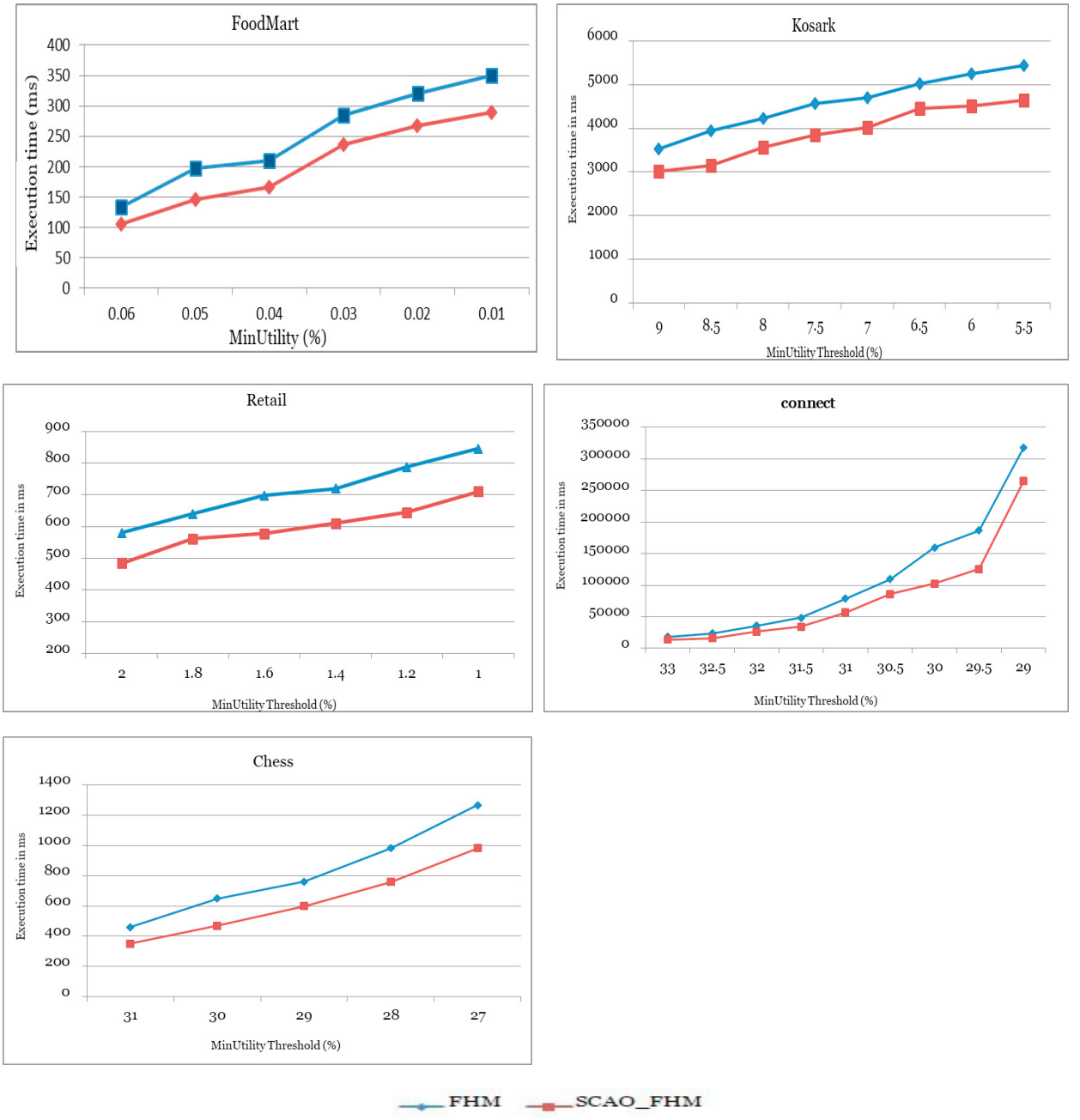

Fig.6. Running time of FHM Vs SCAO_FHM

6. Conclusion

HUIM is commonly used to create decision support systems in various applications. For high utility mining, utility list-based techniques provide more outstanding performance. The performance of the algorithms in terms of execution time depends on utility list join operations. On the other hand, utility list-based algorithms undertake time-consuming utility list join operations. The cost of the utility list join operations is directly related to the number of comparisons. By reducing the number of comparisons, the utility list joins operation cost can be minimized. Proposed an efficient search space exploration sequence as support count ascending order is used to decrease the comparison counts that are required to join utility lists. As a result, the cost in terms of the running time of utility list joins operations is reduced. The time complexity analysis of the proposed approach shows that the presented strategy

minimizes the number of comparisons by exploring the search space in ascending order of support count, enhancing the algorithm's execution time performance.

Moreover, various experiments have been conducted on some standard real datasets to assess the performance of the presented approach. The result shows that the proposed approach 4 to 16 percent, 6 to 18 percent, and 10 to 24 percent faster than HUI-Miner, mHUI-Miner, and ULB-Miner, respectively. This work is carried out on a static dataset. In the future, the possibility of incremental and stream data environments may also be explored. The work is expanded to incremental and stream data environments. The intra-correlation of items within the itemset may be explored.

References

[1] Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB (Vol. 1215, pp. 487-499).

[2] Shen, Y. D., Zhang, Z., & Yang, Q. (2002). Objective-oriented utility-based association mining. In 2002 IEEE International Conference on Data Mining, 2002. Proceedings. (pp. 426-433). IEEE.

[3] H Hamilton, B., & Yao, H. J. (2004). A foundational approach to mining itemset utilitiesfrom databases. In Proceedings Of the 4th SIAM ICDM (pp. 482-486).

[4] Yao, H. (2006). Yao H., Hamilton HJ. Mining itemset utilities from transaction databases, Data & Knowledge Engineering, 59(3), 603-626.

[5] Liu, Y., Liao, W. K., & Choudhary, A. (2005). A two-phase algorithm for fast discovery of high utility itemsets. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 689-695). Springer, Berlin, Heidelberg.

[6] Hu, J., & Mojsilovic, A. (2007). High-utility pattern mining: A method for discovery of high-utility item sets. Pattern Recognition, 40(11), 3317-3324.

[7] Ahmed, C. F., Tanbeer, S. K., Jeong, B. S., & Lee, Y. K. (2009). An efficient candidate pruning technique for high utility pattern mining. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 749-756). Springer, Berlin, Heidelberg.

[8] Tseng, V. S., Wu, C. W., Shie, B. E., & Yu, P. S. (2010). 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 (pp. 253-262).

[9] Tseng, S., & Wu, Y. (2013). Tseng VS, Shie B.-E., Wu C.-W., Yu PS. Efficient algorithms for mining high utility itemsets from transactional databases, IEEE Transactions on Knowledge and Data Engineering, 25(8), 1772-1786.

[10] Song, W., Liu, Y., & Li, J. (2014). Mining high utility itemsets by dynamically pruning the tree structure. Applied intelligence, 40(1), 29-43.

[11] Ryang, H., Yun, U., & Ryu, K. H. (2016). Fast algorithm for high utility pattern mining with the sum of item quantities. Intelligent Data Analysis, 20(2), 395-415.

[12] Qu, J. F., Liu, M., & Fournier-Viger, P. (2019). Efficient algorithms for high utility itemset mining without candidate generation. In High-Utility Pattern Mining (pp. 131-160). Springer, Cham.

[13] Fournier-Viger, P., Wu, C. W., Zida, S., & Tseng, V. S. (2014). FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In International symposium on methodologies for intelligent systems (pp. 83-92). Springer, Cham.

[14] Christen, P., Schnell, R., Vatsalan, D., & Ranbaduge, T. (2017). Efficient cryptanalysis of bloom filters for privacy-preserving record linkage. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 628-640). Springer, Cham.

[15] Duong, Q. H., Fournier-Viger, P., Ramampiaro, H., Nørvåg, K., & Dam, T. L. (2018). Efficient high utility itemset mining using buffered utility-lists. Applied Intelligence, 48(7), 1859-1877.

[16] Liu, M., & Qu, J. (2012). Mining high utility itemsets without candidate generation Proceedings of the 21st ACM international conference on information and knowledge management. ACM, New York, NY, USA, CIKM, 12.

[17] 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.