





Mining Skyline Frequent-Utility Itemsets with Utility Filtering

Wei Song¹ , Chuanlong Zheng¹, and Philippe Fournier-Viger² 

- ¹ School of Information Science and Technology, North China University of Technology, Beijing 100144, China
songwei@ncut.edu.cn
- ² School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China

Abstract. Skyline frequent-utility itemsets (SFUIs) can provide more actionable information for decision-making with both frequency and utility considered. In this paper, the problem of mining SFUIs by filtering utilities from different perspectives is studied. First, filtering by frequency is considered. The max utility array (MUA) structure is designed, which is proved to have a size no larger than the size of arrays in state-of-the-art algorithms. Using the MUA, the utility-list is verified to prune unpromising itemsets and their extensions. Second, filtering using transaction-weighted utilization is applied. The minimum utility of SFUIs is proposed and the proof that this concept can be used as a pruning strategy in the early stage of search space traversal is provided. Finally, filtering using utility itself is also considered. The minimum utility of extension is presented, and its use as a pruning strategy during the extension stage of search space traversal is validated. Based on these filtering methods, a novel algorithm called SFUIs mining based on utility filtering (SFUI-UF) is proposed. Extensive experimental results show that the SFUI-UF algorithm can discover all correct SFUIs with high efficiency and low memory usage.

Keywords: Skyline frequent-utility itemsets · Max utility array · Minimum utility of SFUIs · Minimum utility of extension

1 Introduction

Itemset mining is a core task in data mining. Several types of itemsets [10] have been proposed, among which, frequent itemsets (FIs) and high utility itemsets (HUIs) are the two most influential. FIs are itemsets that have high frequency [6], whereas HUIs are itemsets that have high profit [11].

To meet users' multiple needs and provide them with more informative knowledge, researchers have been paying increasing attention to combining FIs and HUIs. In [12], the utility-frequent itemset (UFI) and its mining algorithm were proposed considering both utility and support. Setting a single threshold of FIs or HUIs is a difficult problem, let alone two thresholds that are needed simultaneously. Furthermore, the algorithm's

efficiency is another challenging issue for mining UFIs because the downward closure property does not hold when support and utility are considered together.

To solve this problem, skyline frequent-utility itemsets (SFUIs) were proposed [2]. An itemset is an SFUI if it is not dominated by any other itemset in the database considering both frequency and utility. Using both frequency and utility is meaningful. For example, parents of students may choose to rent houses for the students according to the price and distance of the houses from the school to save the time that it takes for students to travel to and from school. In this case, the distance from the house to the school and the price of the house are contrasted; that is, a house that is close to the school normally has a higher price than a house that is far from the school.

In the first SKYMINE algorithm [2], a tree structure was used to transform the original information. To improve efficiency, the utility-list (UL) structure was used in SFU-Miner [8], SKYFUP-D [3], and SKYFUP-B [3] algorithms. Because mining SFUIs does not require either a support threshold or utility threshold, the algorithm's efficiency is still the most challenging issue for the SFUI mining (SFUIM) problem.

We propose an SFUIM algorithm called SFUI-UF. First, we design the max utility array (MUA) structure, which is smaller than arrays in mainstream algorithms. Using this array, we propose a UL-based pruning strategy to omit unpromising itemsets and their extensions. Second, we propose the minimum utility of SFUIs (MUS), which is the highest utility of single items that have the highest frequency. Considering transaction-weighted utilization, the MUS is an effective strategy that can prune unpromising single items and all their extensions. Third, we present the minimum utility of extension (MUE), which is the utility of the itemset that generates the enumeration itemsets. The MUE is an effective pruning strategy for search space traversal. Finally, we conducted extensive experiments. Our results showed that the proposed algorithm is efficient and memory-saving compared with mainstream algorithms.

2 SFUIM Problem

Let $I = \{i_1, i_2, \dots, i_m\}$ be a finite set of items. Set $X \subseteq I$ is called an *itemset*, or a k -itemset if it contains k items. Let $D = \{T_1, T_2, \dots, T_n\}$ be a transaction database. Each transaction $T_d \in D$ ($1 \leq d \leq n$), where d is a unique identifier, is a subset of I . The *frequency* of itemset X , denoted by $f(X)$, is defined as the number of transactions in which X occurs as a subset.

The *internal utility* $q(i_p, T_d)$ represents the quantity of item i_p in transaction T_d . The *external utility* $p(i_p)$ is the unit profit value of item i_p . The *utility* of item i_p in transaction T_d is defined as $u(i_p, T_d) = p(i_p) \times q(i_p, T_d)$. The utility of itemset X in transaction T_d is defined as $u(X, T_d) = \sum_{i_p \in X} u(i_p, T_d)$. The utility of itemset X in D is defined as

$u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$. The *transaction utility* (TU) of transaction T_d is defined

as $TU(T_d) = u(T_d, T_d)$. Because utility is not a measure that maintains the downward closure property, Liu et al. [5] proposed the *transaction-weighted utilization* (TWU) model. TWU can be used as the upper bound of utility, and it is proved that an itemset is not an HUI if its TWU is lower than the minimum utility threshold. Formally, the TWU

of itemset X is the sum of the TUs of all the transactions containing X , which is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in \mathbf{D}} TU(T_d)$.

Considering both frequency and utility factors, an itemset X *dominates* another itemset Y in \mathbf{D} , if $f(X) \geq f(Y)$ and $u(X) > u(Y)$, or $f(X) > f(Y)$ and $u(X) \geq u(Y)$, which is denoted by $X \succ Y$. An itemset X in a database \mathbf{D} is an SFUI if it is not dominated by any other itemsets in the database considering both frequency and utility factors. An itemset X is considered as a *potential SFUI* (PSFUI) if there is no itemset Y such that $f(Y) = f(X)$, and $u(Y) > u(X)$, and it was proved that all SFUIs are PSFUIs [8]; hence, we can enumerate the PSFUIs first, and then filter the actual SFUIs from the PSFUIs.

Based on the above definitions, the SFUIM problem is to discover all the non-dominated itemsets in the database by considering both frequency and utility factors.

Table 1. Example database

TID	Transactions	TU
1	(B, 1), (D, 1)	7
2	(A, 1), (B, 1), (C, 1), (D, 1), (E, 1)	18
3	(B, 2), (D, 1), (E, 1)	13
4	(D, 1), (E, 1)	9
5	(A, 4), (B, 1), (D, 1), (E, 2)	39
6	(B, 4), (E,1)	12
7	(C, 2)	2

Table 2. Profit table

Item	A	B	C	D	E
Profit	6	2	1	5	4

As a running example, consider the transaction database in Table 1 and the profit table in Table 2. For convenience, an itemset {D, E} is denoted by DE. In the example database, the utility of item E in transaction T_2 is $u(E, T_2) = 4 \times 1 = 4$, the utility of itemset DE in transaction T_2 is $u(DE, T_2) = 5 + 4 = 9$, and the utility of itemset DE in the transaction database is $u(DE) = u(DE, T_2) + u(DE, T_3) + u(DE, T_4) + u(DE, T_5) = 40$. The TU of T_2 is $TU(T_2) = u(ABCDE, T_2) = 18$, and the utilities of the other transactions are shown in the third column of Table 1. Because DE is contained by transactions T_2, T_3, T_4 , and $T_5, f(DE) = 4$, and $TWU(DE) = TU(T_2) + TU(T_3) + TU(T_4) + TU(T_5) = 79$. Because an itemset that has a frequency of 4 and utility greater than 40 does not exist, DE is considered as a PSFUI. We can further verify that DE is not dominated by any other itemsets, so it is an SFUI.

3 Related Work

Yeh et al. [12] first proposed the concept of UFIs and designed a two-phase algorithm for mining UFIs. Then, the FUFM algorithm was proposed to mine UFIs more efficiently [9]. Both frequency and utility thresholds are required to discover UFIs. However, it is difficult for non-expert users to set these two appropriate thresholds.

To consider both frequency and utility without setting thresholds, Goyal et al. introduced SFUIs and proposed the SKYMINE algorithm for mining them [2]. Considering both frequency and utility, SFUIs are itemsets that are not dominated by any other itemsets. Thus, neither the frequency threshold nor utility threshold is needed. The UP-tree structure is used in the SKYMINE algorithm to transform the original data for mining SFUIs using a pattern-growth approach.

Using the UL structure [4], Pan et al. proposed the SFU-Miner algorithm to discover SFUIs [8]. Compared with the UP-tree, the UL can identify SFUIs directly without candidates. Furthermore, the umax array has been proposed to store information about the maximal utility for each occurrence frequency, which can be used to identify the non-dominated itemsets efficiently based on the utility and frequency measures. With the above structures, the SFU-Miner is more efficient than SKYMINE algorithm.

Lin et al. proposed two algorithms, that is, SKYFUP-D and SKYFUP-B, for mining SFUIs by depth-first and breadth-first traversing the search space, respectively [3]. In these two algorithms, the UL is also used, and the utility-max structure is designed to keep the maximal utility among the itemsets if their frequency is no lower than the index parameter.

Different from the UL, Nguyen et al. designed an extent utility list (EUL) structure in their FSKYMINE algorithm for mining SFUIs [7]. For EUL, the utility of the expanded itemset in UL is decomposed into two fields: the utility of the itemset before extension and the utility of the item before appending to the enumeration itemset. Using EUL, new pruning strategy is proposed to speed up the mining process of SFUIs.

Compared with existing methods, we design new array structure with frequency, and propose three utility filter strategies to prune the search space. Thus, the overall performance is improved.

4 Proposed Algorithm

4.1 UL Structure

The same as the methods in [3, 8], we also use the UL to transform the original data. Let \triangleright be the total order (e.g., lexicographic order) of items in the database \mathbf{D} . The UL of an itemset X , denoted by $X.UL$, is a set of triads, in which each triad consists of three fields denoted by $(tid, iutil, rutil)$, where tid is the ID of a transaction containing X ; $iutil$ is the utility of X in T_{tid} , that is, $iutil(X, T_{tid}) = u(X, T_{tid})$; and $rutil$ is the remaining utility of all the items after X in T_{tid} . Let T_{tid} / X be the set of items after all items of X in T_{tid} according to \triangleright , $rutil(X, T_{tid}) = \sum_{i \in (T_{tid} / X)} u(i, T_{tid})$.

4.2 Max Utility Array

We propose an array to store utility with respect to frequency.

Definition 1 (Max utility array). Let f_{max} be the maximal frequency of all 1-itemsets in \mathbf{D} . The MUA is an array that contains f_{max} elements. The element that has frequency f ($1 \leq f \leq f_{max}$) is defined as

$$\text{MUA}(f) = \max \{ u(X) \mid f(X) \geq f \}, \quad (1)$$

where X is an itemset. The MUA is similar to utilmax [3]; the difference between them is the size of the array. According to [3], the size of utilmax is $|\mathbf{D}|$, that is, the number of transactions in \mathbf{D} , whereas the size of MUA is f_{max} .

Theorem 1. For the same transaction database \mathbf{D} , the MUA covers all frequencies and $|\text{MUA}| \leq |\text{utilmax}|$, where $|\text{MUA}|$ and $|\text{utilmax}|$ are the number of elements in the MUA and utilmax, respectively.

Proof. The frequency measure satisfies the downward closure property; that is, for any two itemsets X and Y , if $X \subseteq Y$, $f(X) \geq f(Y)$.

We first prove that the maximal frequency of 1-itemsets, f_{max} , is also the maximal frequency of all the itemsets. We prove this by contradiction. Let $i_x i_y$ be a 2-itemset that has the highest frequency of all the itemsets. Because $i_x \subset i_x i_y$, $f(i_x) \geq f(i_x i_y)$. Because f_{max} is the maximal frequency of 1-itemsets, $f_{max} \geq f(i_x) \geq f(i_x i_y)$. This contradicts the assumption. Similarly, for any k -itemset Z , we also have $f_{max} \geq f(Z)$. Thus, f_{max} is also the maximal frequency of all the itemsets.

According to Definition 1, all frequencies between 1 and f_{max} correspond to an element of the MUA. Because f_{max} is also the maximal frequency of all the itemsets, the MUA covers all frequencies in \mathbf{D} . Furthermore, $|\text{MUA}| = f_{max} \leq |\mathbf{D}| = |\text{utilmax}|$. \square

In real-world transaction databases, very few itemsets appear in every transaction. Thus, the size of the MUA is no larger than that of utilmax, in most cases, and memory consumption can be saved.

Theorem 2. Let X be an itemset. If the sum of all iutil and rutil values in $X.\text{UL}$ is less than $\text{MUA}(f(X))$, then X and all the extensions of X are not SFUIs.

Proof. According to the assumption, there exists an itemset Y such that $u(Y) = \text{MUA}(f(X))$; that is, $u(Y) > \sum_{d \in X.\text{tids}} (iutil(X, T_d) + rutil(X, T_d))$, where $X.\text{tids}$ denotes the set of tids in $X.\text{UL}$. Thus,

$$u(Y) > \sum_{d \in X.\text{tids}} iutil(X, T_d) = u(X). \quad (2)$$

Based on Definition 1, $f(Y) \geq f(X)$. Thus, Y dominates X and X is not an SFUI.

Let Z be arbitrary extension of X . Then $X \subset Z$; hence, $f(Z) \leq f(X)$. Because $f(X) \leq f(Y)$, $f(Z) \leq f(Y)$ holds. Similarly,

$$u(Y) > \sum_{d \in X.\text{tids}} (iutil(X, T_d) + rutil(X, T_d)) \geq u(Z). \quad (3)$$

Thus, Y also dominates Z and Z is not an SFUI. \square

Based on Theorem 2, we can prune the search space effectively using MUA.

4.3 Pruning Strategies

We also propose two pruning strategies. They are used in the initial stage and extension stage of the SFUI-UF algorithm.

Definition 2 (Minimum utility of SFUIs). Let \mathbf{D} be a transaction database. The minimum utility of SFUIs (MUS) in \mathbf{D} is the maximal utility of single items that have the highest frequency, which is defined as

$$\text{MUS} = \max \{u(i) \mid f(i) = f_{\max}\}, \quad (4)$$

where i is an item in \mathbf{D} and f_{\max} is the maximal frequency of all 1-itemsets in \mathbf{D} .

In the running example, three items have the highest frequency: $f(\text{B}) = 5, f(\text{D}) = 5,$ and $f(\text{E}) = 5$. Since $u(\text{B}) = 18, u(\text{D}) = 25,$ and $u(\text{E}) = 24,$ $\text{MUS} = 25$.

Theorem 3. Let i_x be a 1-itemset. If $\text{TWU}(i_x) < \text{MUS}$, i_x and all itemsets containing i_x are not SFUIs.

Proof. Let i_c be the 1-itemset with $f(i_c) = f_{\max}$ and $u(i_c) = \text{MUS}$. Because $u(i_x) \leq \text{TWU}(i_x) < \text{MUS} = u(i_c)$ and $f(i_x) \leq f_{\max} = f(i_c)$, i_x is dominated by i_c . Thus, i_x is not an SFUI.

Consider an arbitrary itemset X containing i_x : $u(X) \leq \text{TWU}(i_x) < \text{MUS} = u(i_c)$, and $f(X) \leq f(i_x) \leq f_{\max} = f(i_c)$. Thus, X is dominated by i_c and X is not an SFUI. \square

Using Theorem 3, once a 1-itemset is found to have TWU lower than the MUS, this itemset and all its supersets can be pruned safely. In our algorithm, the MUS is set to the initial values of each element in the MUA, whereas for `umax` [8] and `utilmax` [3], all initial values are set to zero.

Definition 3 (Minimum utility of extension). Let X be an itemset, i_x be an item, and $X \cup i_x$ be an extension of itemset X . Then $u(X)$ is defined as the minimum utility of extension (MUE) of $X \cup i_x$, which is denoted by $\text{MUE}(X \cup i_x)$.

Consider itemset A and its extension AD in the running example: $\text{MUE}(AD) = u(A) = 30$.

Theorem 4. Let X be an itemset and Y be an extension of X . If $u(Y) < \text{MUE}(Y)$, Y is not an SFUI.

Proof. Because Y is an extension of X , $X \subset Y$. Hence, $f(Y) \leq f(X)$. Because $u(Y) < \text{MUE}(Y) = u(X)$, Y is dominated by X . Thus, Y is not an SFUI. \square

Using Theorem 4, we can prune those unpromising extensions during search space traversal.

4.4 Algorithm Description

Algorithm 1 describes the proposed SFUI-UF algorithm.

The transaction database is first scanned once to determine the maximal frequency of all 1-itemsets and the MUS (Step 1). According to Theorem 3, Step 2 prunes unpromising items using the MUS, and then the TWU values of the remaining items are updated in

Step 3. The loop from Step 4 to Step 9 sorts the items in ascending order of TWU in each transaction and builds the UL of each item. The next loop (Steps 10–12) initializes each element of the MUA as the MUS. S-PSFUI and S-SFUI denote the sets of PSFUIs and SFUIs, and they are initialized as empty sets in Steps 13 and 14, respectively. The procedure P-Miner (described in Algorithm 2), which determines all PSFUIs, is called in Step 15. The function $tail(X)$ is the set of items after all items in X according to the total order of items, denoted by \triangleright . Specifically, $tail(X) = \{j \in I \mid \text{for } \forall i \in X, i \triangleright j\}$. In the SFUI-UF algorithm, ascending order of TWU is used. Then Step 16 calls the function S-Miner (described in Algorithm 3) to return all SFUIs. Finally, Step 17 outputs all the discovered SFUIs.

Algorithm 1	SFUI-UF
Input	Transaction database D
Output	All SFUIs
1	Scan D once to calculate f_{max} and MUS;
2	Delete items with TWU values lower than MUS;
3	Calculate the TWU values of the remaining items;
4	for each transaction $T_d \in D$ do
5	Sort items in T_d using TWU-ascending order;
6	for each item $i \in T_d$ in D do
7	Update i .UL;
8	end for
9	end for
10	for $j=1$ to f_{max} do
11	MUA(j) = MUS;
12	end for
13	S-PSFUI = \emptyset ;
14	S-SFUI = \emptyset ;
15	S-PSFUI \leftarrow P-Miner(\emptyset , $tail(\emptyset)$, MUA);
16	S-SFUI \leftarrow S-Miner(S-PSFUI);
17	Output all SFUIs.

Algorithm 2 generates the PSFUIs by extending the enumerating itemset via depth-first search space traversal. For each item i in $tail(X)$, Step 2 first formulates a candidate itemset by appending i after X . According to Theorem 4, the new candidate is not processed until its utility is no lower than the MUE (Steps 3–12). If the utility of the candidate is no lower than the corresponding element in the MUA, the MUA is updated using the utility and frequency of the candidate in Step 5, and the candidate is determined as a PSFUI in Step 6. When a new itemset is inserted, some itemsets already in S-PSFUIs may be dominated by the newly inserted itemset. The dominated itemsets are removed from S-PSFUIs in Step 7. If the sum of all the $iutil$ and $rutil$ values in the UL of the new candidate is no lower than the corresponding value in the MUA, the P-Miner procedure is called recursively to determine the new PSFUIs in Step 10. Finally, the updated S-PSFUI is returned in Step 14.

Algorithm 2	P-Miner ($X, tail(X), MUA$)
Input	an itemset X , set of items can be extended $tail(X)$, MUA
Output	PSFUIs generated by appending an item to X
1	for each item $i \in tail(X)$ do
2	$X' = X \cup i$;
3	if $u(X') \geq MUE(X')$ then
4	if $u(X') \geq MUA(f(X'))$ then
5	Update MUA according to $f(X')$ and $u(X')$;
6	S-PSFUI $\leftarrow X'$;
7	remove Y from S-PSFUI if $(f(Y) == f(X'))$;
8	end if
9	if $\sum_{d \in X.tids} (iutil(X', T_d) + rutil(X', T_d)) \geq MUA(f(X'))$ then
10	S-PSFUI \leftarrow P-Miner ($X', tail(X'), MUA$);
11	end if
12	end if
13	end for
14	return S-PSFUI.

In Algorithm 3, all itemsets in the S-PSFUI are checked individually. Only those itemsets that are not dominated by other PSFUIs are verified to be the actual SFUIs.

Algorithm 3	S-Miner (S-PSFUI)
Input	S-PSFUI
Output	All SFUIs
1	for each itemset X in S-PSFUI do
2	for each itemset Y in S-PSFUI do
3	if $(f(X) \geq f(Y) \text{ and } u(X) > u(Y))$ or $(f(X) > f(Y) \text{ and } u(X) \geq u(Y))$ then
4	remove Y from S-PSFUI;
5	end if
6	end for
7	end for
8	S-SFUI \leftarrow S-PSFUI;
9	return S-SFUI.

4.5 Illustrated Example

We use the transaction database in Table 1 and profit table in Table 2 to explain the basic idea of the SFUI-UF algorithm. When D is scanned once, $f_{max} = 5$ and $MUS = 25$. Because $TWU(C) = 20 < MUS$, C is deleted from the database. Then, the TWU values of remaining items are recalculated, and the results are $TWU(A) = 56$, $TWU(B) = 88$, $TWU(D) = 85$, and $TWU(E) = 90$.

Thus, according to TWU -ascending order, the remaining four items are sorted as $A \triangleright D \triangleright B \triangleright E$. With this order, each transaction in the database is reorganized, and the result is shown in Table 3.

Table 3. Reorganized database

TID	Transactions	TU
1	(D, 1), (B, 1)	7
2	(A, 1), (D, 1), (B, 1), (E, 1)	17
3	(D, 1), (B, 2), (E, 1)	13
4	(D, 1), (E, 1)	9
5	(A, 4), (D, 1), (B, 1), (E, 2)	39
6	(B, 4), (E,1)	12

Then, the constructed UL structures for all remaining 1-itemsets are shown in Fig. 1. The MUA contains five elements, and each element is initialized as 25.

We take A as an example. We can calculate $u(A) = 30$ by summing up its *iutils* in Fig. 1. Furthermore, $f(A) = 2$ and $u(A) > MUA(2)$. Then $MUA(2)$ is updated to 30. Accordingly, $MUA(1)$ is also updated to 30. At this moment, $MUA = \{30, 30, 25, 25, 25\}$. Then, A is added into the S-PSFUI as a PSFUI. Because $iutil(A, T_2) + rutil(A, T_2) + iutil(A, T_5) + rutil(A, T_5) = 56 > MUA(2)$, the procedure P-Miner is called recursively taking A, $tail(A) = DBE$, and the MUA as parameters.

A			D			B			E		
tid	iutil	rutil	tid	iutil	rutil	tid	iutil	rutil	tid	iutil	rutil
2	6	11	1	5	2	1	2	0	2	4	0
5	24	15	2	5	6	2	2	4	3	4	0
			3	5	8	3	4	4	4	4	0
			4	5	4	5	2	8	5	8	0
			5	5	10	6	8	4	6	4	0

Fig. 1. ULs of the remaining 1-itemsets

Then, we can obtain a new candidate AD by extending A using D. Similarly, $f(AD) = 2$ and $u(AD) = 40$. Because $u(AD) > MUA(2)$, the MUA is updated to $\{40, 40, 25, 25, 25\}$. Then, AD is added into the S-PSFUI as a PSFUI. Because itemset A is already in the S-PSFUI, and $f(A) = f(AD)$, A is then deleted from the S-PSFUI.

We perform these operations recursively until all items are processed. Finally, $MUA = \{56, 56, 40, 40, 25\}$, and the discovered SFUIs are shown in Table 4. The search space of the running example is shown in Fig. 2.

In Fig. 2, the blue nodes containing a red cross are pruned. C and all its child nodes are pruned according to Theorem 3, whereas ABE is pruned according to Theorem 2.

Table 4. Discovered SFUIs

SFUI	Frequency	Utility
ADBE	2	56
DE	4	40
D	5	25

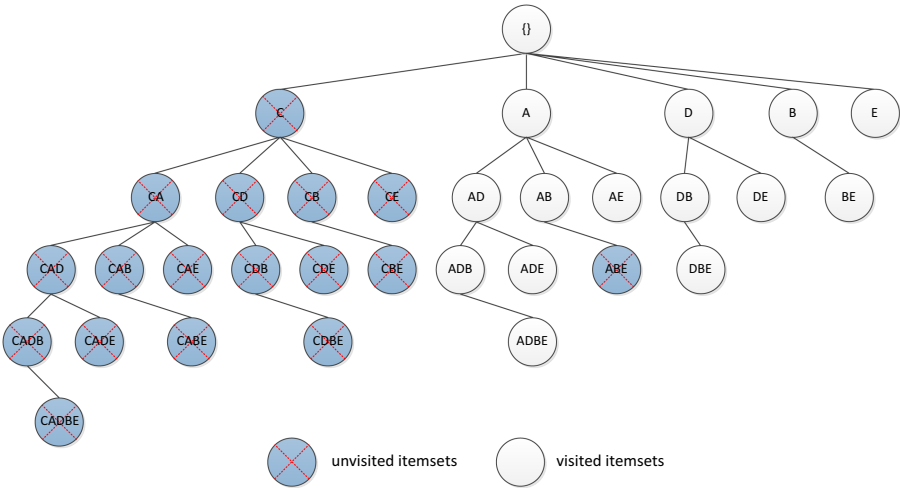


Fig. 2. Search space of the running example

5 Performance Evaluation

In this section, we evaluate the performance of our SFUI-UF algorithm and compare it with SFU-Miner [8], SKYFUP-D [3], and SKYFUP-B [3]. The source code of SFU-Miner was downloaded from the SPMF data mining library [1], and the source codes of SKYFUP-D and SKYFUP-B were provided by the author.

Our experiments were performed on a computer with a 4-Core 3.40 GHz CPU and 8 GB memory running 64-bit Microsoft Windows 10. Our programs were written in Java. We used six datasets for performance evaluation, five real datasets downloaded from the SPMF data mining library [1], and one synthetic dataset generated using the transaction utility database generator provided on the SPMF[1]. We present the characteristics of the datasets in Table 5.

The five real datasets, Chess, Foodmart, Mushroom, Ecommerce, and Connect, contain utility information. The synthetic dataset T25I100D50k do not contain the utility value or quantity of each item in each transaction. Using a random transaction database generator provided in SPMF [1], we generated the unit profit for items following a Gaussian distribution.

Table 5. Characteristics of the datasets

Dataset	#Trans	#Items	Avg. Trans. Len	Max. Trans. Len
Chess	3,196	76	37	37
Foodmart	4,141	1,559	4.42	14
Mushroom	8,124	120	23	23
Ecommerce	14,975	3,803	15.4	29
Connect	67,557	129	43	43
T25I100D50k	50,000	100	25	25

5.1 Search Space Size

We first compare the sizes of the search spaces of the four algorithms, where the size of an algorithm’s search space refers to the number of nodes this algorithm visited. We show the results in Table 6.

Table 6. Comparison of the sizes of the search spaces

Dataset	#SFUIs	Sizes of search spaces			
		SFU-Miner	SKYFUP-D	SKYFUP-B	SFUI-UF
Chess	35	15,433,601	3,750,820	122,363	915,101
Foodmart	1	1,307,831	1,232,033	773,580	773,400
Mushroom	17	810,440	79,710	9,910	22,710
Ecommerce	2	22,309,811	18,590,045	3,471	2,731
Connect	46	-	13,335,429	-	4,349,427
T25I100D50k	6	193,981,690	2,123,115	-	1,595,532

For the six datasets, the proposed SFUI-UF algorithm and SKYFUP-D algorithm returned correct results in all cases, whereas the other two algorithms could not on the Connect dataset. Furthermore, SKYFUP-B algorithm could also not find any results on T25I100D50k dataset. Specifically, SFU-Miner did not return any results after 12 h, and the problem of memory leakage occurred for the SKYFUP-B algorithm. We use “-” for these three entries.

Table 6 clearly shows that the number of SFUIs is very scarce, which is in sharp contrast to the huge search space. Thus, mining SFUIs is a truly challenging problem. Among these four algorithms, SFU-Miner performed worst. This is mainly because the elements of the umax array only recorded the maximal utility of a specific frequency, whose pruning effect was worse than those of the other two array structures. The search spaces of SFUI-UF were always smaller than those of SFU-Miner and SKYFUP-D. The search spaces of SFUI-UF were comparable to those of SKYFUP-B. This shows that

the MUS was effective for search space pruning. This is because once a top-ranked item was pruned using the MUS, a large number of its child nodes were pruned. Thus, the search space was reduced greatly.

5.2 Runtime

We demonstrate the efficiency of our algorithm and the comparison algorithms. We show the comparison results in Table 7.

For the same reason mentioned in Sect. 5.1, we do not report the runtime of SFU-Miner for the Connect dataset, and the runtimes of SKYFUP-B for the Connect and T25I100D50k datasets. Table 7 shows that SFUI-UF was always more efficient than SFU-Miner and SKYFUP-D. Furthermore, SFUI-UF is demonstrated to be more efficient than SKYFUP-B except for the Ecommerce dataset. More importantly, SFUI-UF successfully discovered SFUIs on all six datasets, whereas SKYFUP-B encountered the problem of memory leakage on two datasets. This shows that, in addition to the MUS, the MUE was effective for avoiding generating unpromising candidates. Thus, efficiency improved.

Table 7. Runtime of the compared algorithms

Unit (Sec)	SFU-Miner	SKYFUP-D	SKYFUP-B	SFUI-UF
Chess	169.50	40.14	37.35	29.28
Foodmart	0.57	0.44	0.46	0.34
Mushroom	9.30	2.10	2.22	1.73
Ecommerce	7.37	5.54	0.39	1.20
Connect	-	6,643.63	-	4,176.86
T25I100D50k	2781.29	139.02	-	108.85

5.3 Memory Consumption

We compare the memory usage of the four algorithms. We measured memory usage using the Java API, and show the results in Table 8. SFUI-UF performed best on most datasets, except Foodmart and Connect. Additionally, besides the reasons mentioned in Sect. 5.1 and Sect. 5.2, the MUA structure was also an important factor for saving memory. As stated in Sect. 4.2, the size of the MUA equals the maximal frequency of the 1-itemset, whereas for umax in SFU-Miner [8], and utilmax in SKYFUP-D and SKYFUP-B [3], their sizes are both equivalent to the number of transactions in the database. To show the effect of the MUA, we compared the sizes of the three arrays, and show the results in Table 9.

Table 8. Memory usage of the compared algorithms

Unit (MB)	SFU-Miner	SKYFUP-D	SKYFUP-B	SFUI-UF
Chess	128.24	154.31	1,754.45	41.30
Foodmart	12.48	12.48	85.45	50.70
Mushroom	52.49	123.02	110.50	22.72
Ecommerce	75.08	80.43	69.52	23.50
Connect	-	409.12	-	574.91
T25I100D50k	155.34	128.69	-	90.91

Table 9. Sizes of arrays of the compared algorithms

Dataset	umax	utilmax	MUA
Chess	3,196	3,196	3,195
Foodmart	4,141	4,141	25
Mushroom	8,124	8,124	8,124
Ecommerce	14,975	14,975	1,104
Connect	67,557	67,557	67,473
T25I100D50k	50,000	50,000	12706

The results in Table 9 show that the sizes umax and utilmax were always equivalent to each other on all the six datasets, whereas the size of the MUA was only equivalent to those of umax and utilmax for the Mushroom dataset. The superiority of the MUA was obvious on the three datasets: Foodmart, Ecommerce and T25I100D50k. Particularly, for Foodmart, the size of the MUA was two orders of magnitude smaller than that of the other two arrays. This is because, for most part, the items in the dataset do not appear in every transaction, the length of the MUA will always be less than or equal to the length of umax and utilmax, and the more sparse the dataset, the more significant the difference will be.

6 Conclusion

In this paper, we solve the SFUIM problem by proposing a novel algorithm called SFUI-UF. Because the frequency measure satisfies the downward closure property, we focused our algorithm on utility, similar to other SFUIM algorithms. The main idea of SFUI-UF is to filter utilities from three perspectives, that is, frequency, TWU, and utility itself, which have been formally proved. The SFUI-UF algorithm uses the UL structure to transform the original database, and traverses the search space in a depth-first manner. The experimental results on both real and synthetic datasets showed that SFUI-UF can discover accurate SFUIs with high efficiency and low memory usage.

Acknowledgments. We would like to thank Prof. Jerry Chun-Wei Lin for providing the source codes of the SKYFUP-D and SKYFUP-B algorithms. This work was partially supported by the National Natural Science Foundation of China (61977001), and Great Wall Scholar Program (CIT&TCD20190305).

References

1. Fournier-Viger, P., et al.: The SPMF open-source data mining library version 2. In: Berendt, B., et al. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9853, pp. 36–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46131-1_8
2. Goyal, V., Sureka, A., Patel, D.: Efficient skyline itemsets mining. In: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering, pp.119–124 (2015)
3. Lin, J.C.-W., Yang, L., Fournier-Viger, P., Hong, T.-P.: Mining of skyline patterns by considering both frequent and utility constraints. *Eng. Appl. Artif. Intell.* **77**, 229–238 (2019)
4. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp.55–64 (2012)
5. Liu, Y., Liao, W.-K., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
6. Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: a 25 years review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **9**(6) (2019)
7. Nguyen, H.M., Phan, A.V., Pham, L.V.: FSKYMINE: a faster algorithm for mining skyline frequent utility itemsets. In: Proceedings of the 6th NAFOSTED Conference on Information and Computer Science, pp.251–255 (2019)
8. Pan, J.-S., Lin, J.C.-W., Yang, L., Fournier-Viger, P., Hong, T.-P.: Efficiently mining of skyline frequent-utility patterns. *Intell. Data Anal.* **21**(6), 1407–1423 (2017)
9. Podpecan, V., Lavrac, N., Kononenko, I.: A fast algorithm for mining utility-frequent itemsets. In: Proceedings of International Workshop on Constraint-Based Mining and Learning, pp. 9–20 (2007)
10. Song, W., Jiang, B., Qiao, Y.: Mining multi-relational high utility itemsets from star schemas. *Intell. Data Anal.* **22**(1), 143–165 (2018)
11. Song, W., Zhang, Z., Li, J.: A high utility itemset mining algorithm based on subsume index. *Knowl. Inf. Syst.* **49**(1), 315–340 (2015). <https://doi.org/10.1007/s10115-015-0900-1>
12. Yeh, J.-S., Li, Y.-C., Chang, C.-C.: Two-phase algorithms for a novel utility-frequent mining model. In: Proceedings of the International Workshops on Emerging Technologies in Knowledge Discovery and Data Mining, pp.433–444 (2007)