SFU-CE: Skyline Frequent-Utility Itemset Discovery Using the Cross-Entropy Method

Wei Song[✉] and Chuanlong Zheng

School of Information Science and Technology, North China University of Technology, Beijing 100144, China
songwei@ncut.edu.cn

Abstract. Considering both frequency and utility, skyline frequent-utility itemsets (SFUIs) increase the amount of actionable information available for decision-making. In this paper, we propose an algorithm for mining SFUIs called skyline frequent-utility mining based on cross-entropy (SFU-CE). We first model the SFUI mining problem using CE with utility as the optimization object. We then propose critical utility pruning for unpromising itemsets with utility and its upper bound. Furthermore, we also design a random mutation strategy to make itemsets within each sample more diverse. We demonstrate the efficiency and accuracy of SFU-CE by comparing it with state-of-the-art algorithms.

Keywords: Skyline frequent-utility itemsets · Cross-entropy · Critical utility · Random mutation

1 Introduction

As an important sub-field of data mining, itemset mining is used to discover interesting and useful patterns in transaction databases. To evaluate the interestingness or usefulness of itemsets, various measures have been proposed, among which frequency and profit are the two most influential measures.

Considering both frequency and profit, the utility-frequent itemset (UFI) was designed and two algorithms were proposed in [14]. For the UFI model, both minimum utility and support are required. However, it is difficult for users to set appropriate thresholds and the model incurs a high computational cost.

To solve this problem, Goyal et al. introduced SFUIs and proposed the SKYMINE algorithm for mining them [4]. SFUIs are itemsets that are not dominated by any other itemsets in terms of both frequency and utility. Thus, neither the frequency threshold nor utility threshold is required. Using both frequency and utility is meaningful in practical decision-making. For example, parents 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 [4], the UP-tree structure was used to transform the original information for mining SFUIs. In the recent algorithms SFU-Miner [8], SKYFUP-D [5], and SKYFUP-B [5], the utility-list (UL) structure was used to improve efficiency. In addition to the UL structure, different arrays, such as the umax array and utilmax array, have also been designed to record the utility and frequency of intermediate results. Because mining SFUIs does not require either a support threshold or utility threshold, the algorithm's efficiency remains the most challenging issue for the SFUI mining (SFUIM) problem.

Inspired by biological and physical phenomena, heuristic methods have been used to discover itemsets within an acceptable time. Typical heuristic methods include the genetic algorithm [2], particle swarm optimization [6], and artificial bee colony algorithm [9]. Recently, the cross-entropy (CE) [11, 13] method has also been used to discover top-k high utility itemsets (HUIs) and has demonstrated high efficiency.

Different from TKU-CE and TKU-CE+ [11, 13] that use CE for mining HUIs, we propose a CE-based algorithm called skyline frequent-utility mining based on cross-entropy (SFU-CE) for mining SFUIs. First, we model the SFUIM problem using the typical CE method so that items within itemsets with higher utilities have more opportunities to appear in the next sample. Thus, the sample can filter itemsets with a lower utility, which tend to be dominated by other itemsets. Second, we focus on the maximal utility of items with the highest frequency among all single items, and propose critical utility pruning. Thus, the search space is narrowed using utility and transaction-weighted utilization. Third, we design a random mutation (RM) strategy to improve the diversity of each sample. Different from the typical CE method that generates a new sample strictly following the probability vector, the RM strategy generates new itemsets according to probabilities higher than 0.5. Experimental results demonstrated that SFU-CE discovered sufficient SFUIs in less time.

2 Preliminaries

2.1 SFUIM Problem

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items. Set $X \subseteq \boldsymbol{I}$ is called an *itemset*, or a k-itemset if it contains k items. Let $\boldsymbol{D} = \{T_1, T_2, \ldots, T_n\}$ be a transaction database. Each transaction $T_d \in \boldsymbol{D}$ ($1 \leq d \leq n$), where d is a unique identifier, is a subset of $\boldsymbol{I}$. 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 $\boldsymbol{D}$ is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} u(X, T_d)$. The *transaction utility* (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$. The *transaction-weighted utilization* (TWU) model is generally used as the upper bound of utility, and it has been proven that an itemset is not an HUI if its TWU is lower than the minimum utility threshold [7]. Formally, the TWU of itemset X is the sum of the TUs of all the transactions containing X, which is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} TU(T_d)$.

An itemset X *dominates* another itemset Y in the database if $f(X) \geq f(Y)$ and $u(X) > u(Y)$, or $f(X) > f(Y)$ and $u(X) \geq u(Y)$. An itemset X is an SFUI if it is not dominated by any other itemsets in $\boldsymbol{D}$ considering both frequency and utility factors. The SFUIM problem is to discover all the non-dominated itemsets in the database.

Table 1. Example database

TID	Transactions	TU
1	(A, 1) (B, 3) (C, 1) (E,3) (G,1)	30
2	(B, 1) (C, 1) (F, 2)	10
3	(B, 2) (C, 1) (D, 1) (E, 2)	20
4	(A, 1) (B, 1) (C, 1) (D, 1) (E, 1) (F, 1) (G,1)	20
5	(D, 2) (E, 2)	10
6	(G,1)	4

Table 2. Profit table

Item	A	B	C	D	E	F	G
Profit	2	5	3	3	2	1	4

As a running example, consider the database in Table 1 and the profit table in Table 2. For convenience, an itemset {B, C} is denoted by BC. In the example database, the utility of item B in transaction T_1 is $u(\text{B}, T_1) = 5 \times 3 = 15$, the utility of itemset BC in transaction T_1 is $u(\text{BC}, T_1) = 15 + 3 = 18$, and the utility of itemset BC in the transaction database is $u(\text{BC}) = u(\text{BC}, T_1) + u(\text{BC}, T_2) + u(\text{BC}, T_3) + u(\text{BC}, T_4) = 47$. The TU of T_1 is $TU(T_1) = u(\text{ABCEG}, T_1) = 30$, and the utilities of the other transactions are shown in the third column of Table 1. Because transactions T_1, T_2, T_3, and T_4 contain BC, $f(\text{BC}) = 4$, and $TWU(\text{BC}) = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) = 80$. We can further verify that BC is not dominated by any other itemsets, so it is an SFUI. The other SFUI in this example is BCE, where $f(\text{BCE}) = 3$ and $u(\text{BCE}) = 51$.

2.2 Cross-Entropy Method

The CE method is an effective heuristic method that can be used for solving difficult combinatorial optimization problems (COPs) [1]. In this paper, we determine the SFUIs following the COP methodology.

Let $Y = (y_1, y_2, \ldots, y_n)$ be an n-dimensional binary vector, that is, the value of y_i ($1 \leq i \leq n$) is either zero or one. The goal of the CE method is to reconstruct the unknown vector Y by maximizing the function $S(X)$ using a random search algorithm:

$$S(X) = n - \sum\nolimits_{j=1}^{n} \left| x_j - y_j \right|. \tag{1}$$

A naive approach to determine Y is to repeatedly generate binary vectors $X = (x_1, x_2, \ldots, x_n)$ until a solution is equal to Y, which leads to $S(X) = n$. Elements of the trial binary vector X, that is, $x_1, x_2, \ldots, x_n$, are independent Bernoulli random variables with success probabilities, and these probabilities comprise a probability vector (PV) $P = (p_1, p_2, \ldots, p_n)$. The CE method for COP consists of creating a sequence of PVs $P_0, P_1, \ldots$ and levels $\gamma_1, \gamma_2, \ldots$ such that the sequence $P_0, P_1, \ldots$ converges to the optimal PV and the sequence $\gamma_1, \gamma_2, \ldots$ converges to the optimal performance.

Initially, $P_0 = (1/2, 1/2, \ldots, 1/2)$. For a sample $X_1, X_2, \ldots, X_N$ of Bernoulli vectors, calculate $S(X_i)$ for all i and sort the elements in descending order of $S(X_i)$. Let γ_t be a ρ sample quantile of the performances, that is,

$$\gamma_t = S(X_{\lceil \rho \times N \rceil}), \tag{2}$$

where $\lceil \rho \times N \rceil$ is the smallest integer that is greater than or equal to $\rho \times N$. Then each element of the probability vector is updated by

$$P_{t,j} = \sum_{i=1}^{N} I_{\{S(X_i) \geq \gamma_t\}} \times I_{\{X_{ij}=1\}} \Big/ \sum_{i=1}^{N} I_{\{S(X_i) \geq \gamma_t\}}, \tag{3}$$

where $j = 1, 2, \ldots, n$, $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, t is the iteration number, and

$$I_E = \begin{cases} 1, & \text{if } E \text{ is true} \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where E is an event.

Equation 3 is used iteratively to update the PV until the stopping criterion is met. There are two possible stopping criteria: γ_t does not change for a number of subsequent iterations or the PV has converged to a binary vector.

3 SFU-CE Algorithm

3.1 Modeling SFUI Discovery Using the CE

We use bitmap [12] in SFU-CE to identify transactions that contain the target itemsets. We calculate the frequency and utility values of the target itemsets efficiently using bitwise operations.

Specifically, SFU-CE uses a *bitmap cover* representation for itemsets. In a bitmap cover, there is one bit for each transaction in the database. If item i appears in transaction T_j, then bit j of the bitmap cover for item i is set to one; otherwise, the bit is set to zero. This naturally extends to itemsets. Let X be an itemset, $Bit(X)$ corresponds to the bitmap cover that represents the transaction set for the itemset X. Let X and Y be two itemsets. $Bit(X \cup Y)$ can be computed as $Bit(X) \cap Bit(Y)$, that is, the bitwise-AND of $Bit(X)$ and $Bit(Y)$.

In this paper, a binary vector representing an itemset is called an *itemset vector* (IV), and its dimensions are the same as those of the PV. The IVs within the kth iteration are determined by the kth PV. Then, the $(k + 1)$th PV is determined by the IVs within the kth iteration according to Eq. 3. This procedure is repeated until the termination conditions are reached.

To discover the SFUIs from the transaction database, we use the utility of the itemset to replace Eq. 1 directly; that is, for an itemset X,

$$S(X) = u(X) \tag{5}$$

In each iteration t, we sort a sample $X_1, X_2, \ldots, X_N$ in descending order of $S(X_i)$ ($1 \leq i \leq N$), and update the sample quantile γ_t and the PV P_t accordingly.

3.2 Critical Utility Pruning

Because the frequency measure follows the downward-closure property, the frequencies of 1-itemsets are typically high, which can be used as a pruning strategy to narrow the search space.

Definition 1. The critical utility of SFUIs (CUS) in transaction database $\boldsymbol{D}$ is the maximal utility of single items that have the highest frequency, and is defined as.

$$\text{CUS} = max\{u(i)\,|f(i) = f_{max}\} \tag{6}$$

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

In the running example, three items have the highest frequency: $f(\text{B}) = 4$, $f(\text{C}) = 4$, and $f(\text{E}) = 4$. Because $u(\text{B}) = 35$, $u(\text{C}) = 12$, and $u(\text{E}) = 16$, CUS = 35.

Theorem 1. Let X be an itemset. If $u(X) <$ CUS, X is not an SFUI.

Proof. Let i_c be the 1-itemset, where $f(i_c) = f_{max}$ and $u(i_c) =$ CUS, and i_x be the 1-itemset such that $i_x \in X$. Then, $f(X) \leq f(i_x)$. Because $f(i_x) \leq f_{max}$, $f(X) \leq f_{max}$ holds. Considering $u(X) <$ CUS, X is dominated by i_c and X is not an SFUI. □

Using Theorem 1, once a 1-itemset is found to have utility lower than the CUS, it can be identified immediately as not an SFUI. As the upper bound of utility, it is obvious that TWU can also be used for pruning with the CUS.

Corollary 1. Let X be an itemset. If $TWU(X) <$ CUS*,* X and all itemsets containing X are not SFUIs.

Proof. Let i_c be the 1-itemset with $f(i_c) = f_{max}$ and $u(i_c) =$ CUS. Similar to Theorem 1, we can easily prove that X is not an SFUI.

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

Using Corollary 1, once a 1-itemset is found to have TWU lower than the CUS, this itemset and all its supersets can be pruned safely. In the running example, because $TWU(\text{F}) = 30 < \text{CUS}$, F is deleted from the database.

3.3 Random Mutation

For the typical CE-based itemset discovery algorithm [11], the IVs strictly rely on the PV. Because of the randomness of CE, the algorithm tends to fall into the local minimum, which leads to it missing the correct SFUIs. To increase the number of correct SFUIs, a mutation (a classical concept in the genetic algorithm) is used in the proposed algorithm. Within each iteration, in addition to the IVs following the PVs, SFU-CE also generates part of the new IVs randomly.

For each iteration t, let $P = (p_1, p_2,\ldots, p_n)$ be the current PV with j $(1 \leq j \leq n)$ probabilities higher than 0.5. These j probabilities are denoted by p_{r1}, p_{r2}, …, p_{rj} $(1 \leq r_1 \leq r_2 \leq \ldots \leq r_j \leq n)$. To perform RM, a positive integer num $(1 \leq num \leq j)$ is generated first, then num bits (among the r_1-th, r_2-th, …, r_j-th bits) of an IV performing RM are selected randomly, and their values are set to one and the other bits are set to zero.

Let α $(0 < \alpha < 1)$ be the *mutation factor*. In each iteration, $\lfloor N \times \alpha \rfloor$ IVs are generated by RM, and the remaining $N - \lfloor N \times \alpha \rfloor$ IVs are still generated according to the PV. It should be noted that only the bits corresponding to probabilities strictly higher than 0.5 perform RM because if the probability of 0.5 is also considered, all the IVs in the first iteration are all generated by RM.

Consider the running example, suppose the PV in one iteration is <0.3, 0.8, 0.7, 0.2, 0.6, 0.1>. There are only six probabilities because F is deleted using Corollary 1. In this PV, three probabilities are higher than 0.5: the second probability, third probability, and fifth probability. To generate an IV using this PV, a positive integer no higher than three is generated first. For example, let this number be two. Then, two bits within the second, third, and fifth bits are set to one, and the other bits are set to zero. We can see that <010010> is such an IV corresponding to itemset BE.

3.4 Algorithm Description

The proposed SFU-CE algorithm for mining SFUIs is described in Algorithm 1.

```
Algorithm 1   SFU-CE
Input         Transaction database $\boldsymbol{D}$, sample numbers $N$, quantile parameter $\rho$,
              maximum number of iterations $max\_iter$, mutation factor $\alpha$
Output        SFUIs
1   Initialization( );
2   while $t \le max\_iter$ and $P_t$ is not a binary vector do
3     Calculate $P_t$ using Eq. 3;
4     Perform RM;
5     Generate the remaining IVs according to $P_t$;
6     Sort the $N$ itemsets in utility-descending order and denote them by $X_1, X_2, \ldots, X_N$;
7     for $i = 1$ to $N$ do
8       $CSFUI$ = SFUI-Filter($X_i$, $CSFUI$);
9     end for
10    Calculate $\gamma_t$ using Eq. 2;
11    $t$ ++;
12  end while
13  Output all SFUIs in $CSFUI$.
```

In Algorithm 1, the procedure Initialization, described in Algorithm 2, is called in Step 1. The loop from Step 2 to Step 12 discovers the SFUIs when the iteration number is no higher than the threshold and the PV is not a binary vector. For a binary PV, all the N itemsets are the same in one iteration because each item in each itemset is definitely one or zero. In Step 3, the PV is updated according to the current sample. Within the current sample, $\lfloor N \times \alpha \rfloor$ IVs are generated by RM (Step 4), whereas other IVs are produced with the probabilities in PV (Step 5). In Step 6, the itemsets are sorted in descending order of utility. In the loop from Step 7 to Step 9, each itemset is checked by the function SFUI-Filter (described in Algorithm 3). Then, in Step 10, the sample quantile is updated, and in Step 11, the iteration number is incremented by one. Finally, all itemsets in *CSFUI* are output as SFUIs in Step 13.

```
Algorithm 2   Initialization( )
1   Scan $\boldsymbol{D}$ once to delete items with TWU values lower than the CUS;
2   Represent the database using a bitmap;
3   $P_0 = (1/2, 1/2, \ldots, 1/2)$;
4   Generate all itemsets of the first iteration with $P_0$;
5   Sort the $N$ itemsets in utility-descending order and denote them by $X_1, X_2, \ldots, X_N$;
6   $CSFUI = \varnothing$;
7   for $i = 1$ to $N$ do
8     $CSFUI$ = SFUI-Filter($X_i$, $CSFUI$);
9   end for
10  Calculate $\gamma_t$ using Eq. 2;
11  $t = 1$;
```

In Algorithm 2, in Step 1, the database is scanned and unpromising items are pruned using Corollary 1. Next, the database is represented by a bitmap in Step 2. In Step 3, all the probabilities in the PV are initialized to 1/2. Using this PV, the itemsets of the first iteration are generated in Step 4. In Step 5, the itemsets are sorted in utility-descending order and in Step 6, *CSFUI*, the set of candidate SFUIs is initialized as an empty set. In the loop from Steps 7 to 9, each itemset is checked by the function SFUI-Filter (described in Algorithm 3). In Step 10, the sample quantile is calculated. Finally, the iteration number is set to one in Step 11.

Algorithm 3 **SFUI-Filter(*X*, CSFUI)**
Input Itemset *X*, set of candidate SFUIs *CSFUI*
Output *CSFUI*

```
1  if $u(X) \geq$ CUS then
2    Remove all itemsets dominated by $X$ in $CSFUI$;
3    if $X$ is dominated by any itemset in $CSFUI$ then
4      return $CSFUI$;
5    end if
6    $X \rightarrow CSFUI$;
7  end if
8  return $CSFUI$;
```

According to Theorem 1, Algorithm 3 does not perform a meaningful operation unless the utility of the enumerating itemset is no lower than the CUS (Steps 1 to 7). In Step 2, the itemsets in *CSFUI* are deleted if it is dominated by the enumerating itemset. If the enumerating itemset is dominated by an itemset in *CSFUI*, in Steps 3 to 5, the algorithm is terminated and *CSFUI* is returned. If the enumerating itemset is not dominated by any itemset in *CSFUI*, it is inserted into *CSFUI* in Step 6. Finally, if the algorithm is not terminated before, *CSFUI* is returned in Step 8.

4 Performance Evaluation

We evaluate the performance of SFU-CE algorithm and compare it with SKYMINE [4], SFU-Miner [8], SKYFUP-D [5], and SKYFUP-B [5]. We downloaded the source code of SKYMINE and SFU-Miner from the SPMF data mining library [3], and obtained the source code of SKYFUP-D and SKYFUP-B from the author.

We performed the experiments on a computer with a quad-core 3.40 GHz CPU and 8 GB memory running 64-bit Microsoft Windows 10. We wrote our programs in Java. We used four datasets for performance evaluation, two synthetic datasets generated using the transaction utility database generator provided on the SPMF [3], and two real datasets downloaded from the SPMF [3]. The characteristics of the datasets are presented in Table 3.

For all experiments, we set the sample size to 2,000, maximum number of iterations to 2,000, quantile ρ to 0.2, and mutation factor α to 0.3.

Table 3. Characteristics of the datasets

Datasets	Avg. trans. Length	No. of items	No. of trans
T25I50D10k	25	50	10,000
T35I10050k	35	100	50,000
Chess	37	75	3,156
Connect	43	129	67,557

4.1 Runtime

First, we consider the efficiency performance of these algorithms, and show the comparison results in Table 4.

Table 4. Execution times of the five algorithms

Unit (Sec)	SKYMINE	SFU-Miner	SKYFUP-D	SKYFUP-B	SFU-CE
T25I50D10k	20.03	1915.78	77.75	–	7.82
T35I10050k	163.67	–	1006.42	–	69.28
Chess	–	167.53	37.92	36.22	16.20
Connect	–	–	6523.72	–	673.28

For the four datasets, the proposed SFU-CE algorithm and SKYFUP-D algorithm returned results in all cases, whereas the other three algorithms could not. Specifically, SKYMINE ran out of memory on two datasets, SKYFUP-B ran out of memory on three datasets, and SFU-Miner did not return any results after 20 h on two datasets. We use "-" for these seven entries.

Table 4 shows that the proposed SFU-CE algorithm always ran faster than the compared algorithms. On T35I100D50k, SFU-CE was one order of magnitude faster than SKYFUP-D. This shows that CE improved the efficiency of SFUIM. There are three reasons for the efficiency improvement. First, different from the four exact algorithms, SFU-CE only considered itemsets that had high utility values with gradual PV convergence and RM rather than all combinations of items. Second, the critical utility pruning strategies using Theorem 1 and Corollary 1 were also effective for omitting unpromising itemsets. Third, SFU-CE did not record the utilities of different frequencies with different array structures, such as umin in SKYMINE [4], umax in SFU-Miner [8], and utilmax in SKYFUP-D and SKYFUP-B [5]. Accordingly, the time cost of dominant relation checking within these arrays was avoided.

4.2 Accuracy

A heuristic SFUIM algorithm cannot ensure the discovery of all the correct itemsets within a certain number of iterations; that is, some itemsets discovered by SFU-CE may not correspond to the actual SFUIs of the entire dataset. So we compare the percentage of SFUIs discovered by SFU-CE to the actual SFUIs. We used the SKYFUP-D algorithm to discover the actual SFUIs from the four datasets, and used the following equation to calculate the accuracy of SFUIs discovered by SFU-CE:

$$Acc = Num\_CE/Num \times 100\%, \tag{7}$$

where *Num_CE* is the number of actual SFUIs discovered by SFU-CE and *Num* is the total number of actual SFUIs.

Table 5. Accuracy for the four datasets

Datasets	*Num_CE*	*Num*	*Acc* (%)
T25I50D10k	6	6	100
T35I10050k	4	4	100
Chess	35	35	100
Connect	42	46	91.30

Table 5 shows that SFU-CE discovered SFUIs with 100% accuracy, except on the Connect dataset, for which the average accuracy was 97.825% on the four datasets. These results demonstrate that the CE-based algorithm discovered most of actual SFUIs in less time.

4.3 Diversity

To verify the effect of RM proposed in Sect. 3.3, we evaluated the degree of diversity of the mining results using the bit edit distance (BED) [10], which is defined as.

$$BED(V, V_t) = NBits, \tag{8}$$

where V and V_t are two IVs and *NBits* is the number of bitwise-complement operations transformed from V to V_t. We can see from Eq. 8 that the greater the value of $BED(V, V_t)$, the more obvious the difference between V and V_t. For example, transforming $V = <110100>$ to $V_t = <101110>$ requires three bitwise-complement operations: transform the second bit from 1 to 0, transform the third bit from 0 to 1, and transform the fifth bit from 0 to 1. Thus, $BED(V, V_t) = 3$.

Following [10], we used two types of BED in our experiments: the greatest degree of diversity of all pairs of IVs, *maximal BED* (Max_BED), and the average degree of diversity of all pairs of IVs, *average BED* (Ave_BED). Let $V_1, V_2, \ldots, V_N$ be the IVs in one iteration. Max_BED is defined as.

$$Max\_BED = max\{BED(V_i, V_j) |\ 1 \le i \le N,\ 1 \le j \le N, i \ne j\}, \tag{9}$$

and Ave_BED is defined as

$$Ave\_BED = \sum_i \sum_{j \ne i} BED(V_i, V_j) \Big/ N \times (N-1) \cdot \tag{10}$$

We also implemented an algorithm that strictly generates all the IVs within each iteration according to the PV, without RM, and refer to this algorithm as SFU-Base. Because the number of iterations for which the two algorithms converged was inconsistent, we used the percentage of the total number of iterations when comparing their diversity.

Table 6. BED for the T25I50D10k dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	4.58	18.0	4.89	18.0
50	2.46	13.0	2.73	15.0
75	1.56	9.0	1.64	9.0
100	0.0	0.0	0.27	1.0

Table 7. BED for the T35I10050k dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	11.82	23.0	10.85	24.0
50	1.97	9.0	2.92	13.0
75	1.06	4.0	1.55	6.0
100	0.0	0.0	0.0	0.0

Tables 6 and 7 show the diversity comparison results on the two synthetic datasets. Although the diversity of SFU-CE was better, its advantage over SFU-Base was not obvious. The reason behind these results is that all SFUIs on these two synthetic datasets were 1-itemset. This means that there was only one probability in the PV that was higher than 0.5; hence, the effect of RM was not obvious.

The diversity comparison results on the two real datasets are shown in Tables 8 and 9. In the first few iterations, the effect was similar to that on the synthetic datasets: the difference between the two algorithms was not obvious. As the number of iterations increased, the diversity advantage of SFU-CE became increasingly significant. The results demonstrated that the proposed RM strategy improved the diversity of the samples. Consequently, the execution speed increased and accuracy improved.

Table 8. BED for the chess dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	13.40	28.0	11.46	28.0
50	5.83	16.0	6.36	16.0
75	1.97	7.0	3.95	13.0
100	0.0	0.0	2.91	12.0

Table 9. BED for the connect dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	12.73	26.0	8.22	24.0
50	7.62	19.0	7.39	19.0
75	2.43	9.0	5.10	18.0
100	0.0	0.0	3.92	16.0

5 Conclusion

In this paper, we studied the SFUIM problem from the perspective of CE and proposed an SFUIM algorithm called SFU-CE. Because SFUIs are itemsets that are not dominated by other itemsets when considering both frequency and utility constraints, we used utility as the optimization object of CE that checked intermediate results using utility, and proposed critical utility pruning that filtered intermediate results by frequency and utility. To improve the diversity of each sample, we designed RM to generate some new itemsets in addition to the PV. Experiments on publicly available datasets demonstrated that the SFU-CE algorithm was efficient, accurate, and produced diverse samples.

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

References

1. de Boer, P.-T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Ann. OR **134**(1), 19–67 (2005)
2. Djenouri, Y., Comuzzi, M.: GA-apriori: combining apriori heuristic and genetic algorithms for solving the frequent itemsets mining problem. In: Kang, U., Lim, E.-P., Yu, J.X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10526, pp. 138–148. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67274-8_13

3. 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
4. Goyal, V., Sureka, A., Patel, D.: Efficient skyline itemsets mining. In: Proceedings of the Eighth International C* Conference on Computer Science and Software Engineering, pp. 119–124 (2015)
5. 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)
6. Lin, J.-W., Yang, L., Fournier-Viger, P., Hong, T.-P., Voznak, M.: A binary PSO approach to mine high-utility itemsets. Soft. Comput. **21**(17), 5103–5121 (2016). https://doi.org/10.1007/s00500-016-2106-1
7. 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
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. Song, W., Huang, C.: Discovering high utility itemsets based on the artificial bee colony algorithm. In: Phung, D., Tseng, V.S., Webb, G.I., Ho, B., Ganji, M., Rashidi, L. (eds.) PAKDD 2018. LNCS (LNAI), vol. 10939, pp. 3–14. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93040-4_1
10. Song, W., Li, J.: Discovering high utility itemsets using set-based particle swarm optimization. In: Yang, X., Wang, C.-D., Islam, M.S., Zhang, Z. (eds.) ADMA 2020. LNCS (LNAI), vol. 12447, pp. 38–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65390-3_4
11. Song, W., Liu, L., Huang, C.: TKU-CE: cross-entropy method for mining top-k high utility itemsets. In: Fujita, H., Fournier-Viger, P., Ali, M., Sasaki, J. (eds.) IEA/AIE 2020. LNCS (LNAI), vol. 12144, pp. 846–857. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55789-8_72
12. Song, W., Liu, Y., Li, J.: Vertical mining for high utility itemsets. In: Proceedings of the 2012 IEEE International Conference on Granular Computing, pp. 429–434 (2012)
13. Song, W., Zheng, C., Huang, C., Liu, L.: Heuristically mining the top-k high-utility itemsets with cross-entropy optimization. Appl. Intell. (2021). https://doi.org/10.1007/s10489-021-02576-z
14. 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)