TKU-CE: Cross-Entropy Method for Mining Top-K High Utility Itemsets

Wei Song [0000-0003-0649-8850], Lu Liu, and Chaomin Huang

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

Abstract. Mining high utility itemsets (HUIs) is one of the most important research topics in data mining because HUIs consider non-binary frequency values of items in transactions and different profit values for each item. However, setting appropriate minimum utility thresholds by trial and error is a tedious process for users. Thus, mining the top-*k* high utility itemsets (top-*k* HUIs) without setting a utility threshold is becoming an alternative to determining all of the HUIs. In this paper, we propose a novel algorithm, named TKU-CE (Top-K high Utility mining based on Cross-Entropy method), for mining top-*k* HUIs. The TKU-CE algorithm follows the roadmap of cross entropy and tackles top-*k* HUI mining using combinatorial optimization. The main idea of TKU-CE is to generate the top-*k* HUIs by gradually updating the probabilities of itemsets with high utility values. Compared with the state-of-the-art algorithms, TKU-CE is not only easy to implement, but also saves computational costs incurred by additional data structures, threshold raising strategies, and pruning strategies. Extensive experimental results show that the TKU-CE algorithm is efficient, memory-saving, and can discover most actual top-*k* HUIs.

Keywords: Data mining, Heuristic method, Cross-entropy, Combinatorial optimization, Top-*k* high utility itemset.

1 Introduction

High utility itemset (HUI) mining [7] is an extension of frequent itemset (FI) mining [1] used to discover high-profit itemsets by considering both the quantity and value of a single item. However, it is difficult for non-expert users to set an appropriate threshold. Consequently, top-*k* high utility itemset (top-*k* HUI) mining [13] is drawing researchers' attention. The *k* value is a more intuitive and direct parameter for users to set than the minimum threshold.

Top-*k* HUI mining uses the same concept of utility as HUI mining; that is, an item's utility is mainly reflected by the product of its profit and occurrence frequency. The *k* itemsets with the highest utility values comprise the target of top-*k* HUI mining. Wu et al. [13] first introduced the problem of top-*k* HUI mining, and proposed the TKU algorithm following the widely used two phase routine in HUI algorithms [7]. Later, they proposed a more efficient algorithm TKO using the one phase routine

[11]. Recently, several other top-*k* HUIs mining algorithms, such as kHMC [3] and TKEH [9], have also been proposed.

Whether top-*k* HUI mining is approached using a two-phase or one-phase algorithm, it is equivalent to HUI mining with the minimum utility threshold set to zero. Thus, the key techniques of these algorithms amount to various threshold raising strategies; that is, the minimum utility threshold increases gradually as the intermediate top-*k* results progress. Thus, in this paper, we aim to use a different strategy that does not require constant threshold-raising. We achieve this goal using the cross-entropy method.

The cross-entropy (CE) method is a combinatorial and multi-extremal optimization approach [2]. The CE method approaches the optimal values using an iterative procedure where each iteration is composed of two phases: generating a random data sample, and updating parameters to produce better samples in the next iteration. According to the number of samples *N*, better results are retained and worse results are abandoned in each iteration. After many cycles of iterations, the best *N* results are obtained. This approach is essentially consistent with the problem of top-*k* HUI mining. Thus, we use the CE method to formulate the novel top-*k* HUI mining algorithm proposed in this paper. The major contributions of this work are summarized as follows:

First, TKU-CE directly uses the utility value as the fitness function, and models the problem of top-*k* HUI mining through the perspective of combinatorial optimization. Second, TKU-CE updates a probability vector gradually. In this probability vector, there is a higher likelihood of updating the probability corresponding to itemsets with higher utility values. The experimental results show that TKU-CE is not only efficient, but also requires less memory resources. Furthermore, TKU-CE can discover more than 90% of the actual top-*k* HUIs in most cases.

2 Preliminaries

2.1 Top-K HUI Mining Problem

Let $\boldsymbol{I}= \{i_1, i_2,\ldots, i_m\}$ be a finite set of items, and $X \subseteq \boldsymbol{I}$ is called an *itemset*. Let $\boldsymbol{D}=\{T_1, T_2, \ldots, T_n\}$ be a transaction database. Each transaction $T_i \in \boldsymbol{D}$, with unique identifier *tid*, is a subset of $\boldsymbol{I}$.

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 $u(X,T_d) = \sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d)$. The utility of itemset X in $\boldsymbol{D}$ is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$. The transaction utility of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$. The *minimum utility threshold* δ is given as a percentage of the total transaction utility values of the database, while the *minimum utility value* is defined as $min\_util = \delta \times \sum_{T_d \in D} TU(T_d)$. An itemset X is called a *high utility itemset* if $u(X) \geq min\_util$. The set of all HUIs in $\boldsymbol{D}$ w.r.t. min_util is denoted by $f_H(\boldsymbol{D}, min\_util)$.

An itemset X is called a top-k HUI in a database $\boldsymbol{D}$ if there are less than k itemsets whose utilities are larger than $u(X)$ in $f_H(\boldsymbol{D}, 0)$. Top-k HUI mining aims to discover the k itemsets with the highest utilities, where k is a parameter set by the user.

Table 1. Example database

TID	Transactions	TU
T_1	(A, 1) (B, 1) (C, 1) (F, 2)	19
T_2	(B, 1) (D, 1) (E, 1)	9
T_3	(A, 1) (B, 1) (C, 1) (F, 1)	15
T_4	(C, 3) (D, 2) (F, 1)	17
T_5	(A, 1) (C, 2)	13

Table 2. Profit table

Item	A	B	C	D	E	F
Profit	7	1	3	2	6	4

*Example 1***.** Consider the database in Table 1 and the profit table in Table 2. For convenience, we write an itemset $\{B, C\}$ as BC. In the example database, the utility of item C in transaction T_1 is: $u(C, T_1) = 3\times1 = 3$, the utility of itemset BC in transaction T_1 is: $u(BC, T_1) = u(B, T_1) + u(C, T_1) = 4$, and the utility of itemset BC in the transaction database is $u(BC) = u(BC, T_1) + u(BC, T_3) = 8$. The transaction utility of T_5 is: $TU(T_5) = u(AC, T_5) = 13$. The utilities of the other transactions are given in the third column of Table 1. In this example, the set of top-3 HUIs is $\{ABCF$: 34, AC: 33, ACF: 32$\}$, where the number beside each itemset indicates its utility.

2.2 Cross-Entropy Method

The CE method can be used either for estimating probabilities of rare events in complex stochastic networks, or for solving difficult combinatorial optimization problems (COP). In this paper, we determine the top-k HUIs following the COP methodology.

The classical CE for COPs involving binary vectors is formalized as follows. Let $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ be an n-dimensional binary vector, that is, the value of y_i ($1 \le i \le n$) is either zero or one. The goal of the CE method is to reconstruct the unknown vector $\boldsymbol{y}$ by maximizing the function $S(\boldsymbol{x})$ using a random search algorithm.

$$S(x) = n - \sum_{j=1}^{n} | x_j - y_j | \tag{1}$$

A naive way to find $\boldsymbol{y}$ is to repeatedly generate binary vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ until a solution is equal to $\boldsymbol{y}$, leading to $S(\boldsymbol{x}) = n$. Elements of the trial binary vector $\boldsymbol{x}$, namely $x_1, x_2, \ldots, x_n$ are independent Bernoulli random variables with success probabilities $p_1, p_2, \ldots, p_n$, and these probabilities can comprise a probability vector $p' = (p_1, p_2, \ldots, p_n)$. The CE method for COP consists of creating a sequence of probability vectors $p'_0, p'_1, \ldots$ and levels $\gamma_1, \gamma_2, \ldots$, such that the sequence $p'_0, p'_1, \ldots$ converges to the optimal probability vector, and the sequence γ_1, γ_2, … converges to the optimal performance.

Initially, $p'_0 = (1/2, 1/2, ..., 1/2)$. For a sample $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, …, $\boldsymbol{x}_N$ of Bernoulli vectors, calculate $S(\boldsymbol{x}_i)$ for all i, and order the elements according to descending $S(\boldsymbol{x}_i)$. Let γ_t be a ρ sample quantile of the performances. That is:

$$\gamma_t = S_{(\lceil \rho \times N \rceil)} \tag{2}$$

Then each element of the probability vector is updated by:

$$p'_{t,j} = \sum_{i=1}^{N} I_{\{S(x_i) \geq \gamma_t\}} I_{\{x_{ij}=1\}} \bigg/ \sum_{i=1}^{N} I_{\{S(x_i) \geq \gamma_t\}} \tag{3}$$

where $j = 1, 2, …, n$, $\boldsymbol{x}_i = (x_{i1}, x_{i2}, …, x_{in})$, t is the iteration number, and $I(\cdot)$ is the indicator function defined as:

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

where E is an event.

Eq. 3 is used iteratively to update the probability vector 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 probability vector has converged to a binary vector.

3 Existing Algorithms

3.1 Top-K HUI Mining

The basic concepts of top-*k* HUI mining were given by Wu et al. [13]. Since the anti-monotonicity-based pruning strategies of top-*k* FI mining [12] cannot be used directly for top-*k* HUI mining, Wu et al. introduced the concept of the optimal minimum utility threshold, and used a threshold raising method to improve the mining efficiency. REPT [8] is another top-*k* HUI mining algorithm that follows the two phase methodology. The algorithm constructs a global tree structure to generate candidate top-*k* HUIs using three threshold raising strategies, and exploits exact and pre-evaluated utilities of itemsets with a length of one or two to reduce the number of candidates.

Recent algorithms focus on mining top-*k* HUIs directly without generating candidates. The TKO algorithm [11] utilizes a utility list data structure to maintain itemset information during the mining process. Furthermore, TKO also uses three pruning strategies to facilitate the mining process. kHMC [3] also mines the top-*k* HUIs in one phase. Besides a utility list, kHMC proposes the concept of coverage to raise the intermediate thresholds.

For existing top-*k* HUI mining algorithms, the major challenge differentiating top-*k* HUI mining and traditional HUI mining is the threshold raising strategies. Gradually raising the minimum utility threshold constricts the search space during the mining process. Thus, new methods that achieve suitable performance without using threshold raising strategies are pertinent for top-*k* HUI mining.

3.2 HUI Mining Using Heuristic Methods

Inspired by biological and physical phenomena, heuristic methods are effective for solving combinatorial problems, and have been used to traverse immense candidate itemset spaces within an acceptable time for mining FIs and HUIs.

Two HUI mining algorithms, HUPE_{UMU}-GARM and $\text{HUPE}_{\text{WUMU}}$-GARM, based on the genetic algorithm (GA) are proposed in [5]. Premature convergence is the main problem of these two algorithms; that is, the two algorithms easily fall into local optima. Particle swarm optimization (PSO) is another heuristic method used for mining HUIs. The PSO-based algorithm discovers HUIs comprehensively using local optimization and global optimization [6].

Unlike GAs and PSOs, ant colony optimization (ACO) produces a feasible solution in a constructive way. Wu et al. proposed an ACO-based algorithm called HUIM-ACS for mining HUIs [14]. This algorithm generates a routing graph before all of the ants start their tours. Furthermore, positive pruning and recursive pruning are used to improve the algorithm's efficiency.

Song and Huang studied the problem of HUI mining from the perspective of the artificial bee colony (ABC) algorithm. The proposed HUIM-ABC discovers HUIs by modeling the itemsets as nectar sources [10]. For each nectar source, three types of bees are used: employed bee, onlooker bee, and scout bee, for sequential optimization within one iteration. During one iteration, the algorithm outputs an itemset when it is verified as an HUI. This process is executed iteratively until the maximal cycle number is reached.

To the best of our knowledge, the heuristic method of cross-entropy has neither been used in HUI mining nor in top-*k* HUI mining.

4 The Proposed TKU-CE Algorithm

4.1 Bitmap Item Information Representation

The first component of the proposed TKU-CE algorithm is the representation of items. We use bitmap, an effective representation of item information in FI mining and HUI mining algorithms, in TKU-CE to identify transactions containing the target itemsets. We can calculate the utility values of the target itemsets efficiently using bitwise operations.

Specifically, TKU-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 $Bit(X) \cap Bit(Y)$, i.e., the bitwise-AND of $Bit(X)$ and $Bit(Y)$.

4.2 Modeling Top-K HUI Mining Based on the CE Method

After transforming the database into a bitmap, it is natural to encode each solution in a binary vector. To discover the top-*k* HUIs 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 $S(\boldsymbol{x}_i)$ ($1 \le i \le N$), and update the sample quantile γ_t and the probability vector p'_t accordingly.

4.3 The Proposed Algorithm

Algorithm 1 describes our top-k HUI mining algorithm TKU-CE.

```
Algorithm 1   TKU-CE
Input         Transaction database D, the number of desired HUIs k, sample numbers N,
              the quantile parameter ρ, the maximum number of iterations max_iter
Output        Top-k high utility itemsets

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      for $i$=1 to $N$ do
5        for $j$=1 to $|\boldsymbol{I}|$ do
6          Generate $X_{ij}$ with the probability of $p'_{t,j}$;
7        end for
8      end for
9      Sort the $N$ itemsets by descending order of utility;
10     Update the set of top-$k$ HUIs $KH$ using the new sample;
11     Calculate $\gamma_t$ using Eq. 2;
12     $t$++;
13   end while
14   Output top-$k$ HUIs.
```

In Algorithm 1, the procedure Initialization( ), described in Algorithm 2, is first called in Step 1. The main loop from Step 2 to Step 13 calculates the top-*k* HUIs iteratively. Besides the maximal number of iterations, the probability vector becoming a binary vector is also a stopping criterion. With a binary probability vector, all of the N itemsets are the same in one iteration, because each item is definitely included or not included in each itemset. For example, there are five items A, B, C, D, and E. After many iterations, the probability vector converges to (1, 1, 1, 0, 0), then the itemsets within the sample are all ABC because the probabilities of the bits corresponding to D

and *E* are zero. Step 3 calculates the probability vector of the new iteration. The loop from Step 4 to Step 8 generates *N* new itemsets bit by bit. Here, $|\boldsymbol{I}|$ is the number of items in $\boldsymbol{I}$. Specifically, for itemset X_i and its *j*th bit, we randomly generate a probability $p_{i,j}$, then determine the value X_{ij} by:

$$X_{ij} = \begin{cases} 1, \text{ if } p_{i,j} \le p'_{t,j} \\ 0, \text{ if } p_{i,j} > p'_{t,j} \end{cases} \tag{6}$$

Step 9 arranges the itemsets in descending order of utility. We update *KH*, the set of the top-*k* HUIs, according to the latest sample in Step 10. The new ρ sample quantile is calculated in Step 11. Step 12 increments the iteration number by one. Finally, Step 14 outputs all of the discovered top-*k* HUIs.

It should be noted that the number of resulting itemsets of top-*k* HUI mining may be either less than *k* or more than *k*. As in TKU and TKO [11], we output the actual results regardless of whether the number of results is more or less than *k*.

```
Algorithm 2   Initialization( )
 1   Represent the database using a bitmap;
 2   $p'_0$ =(1/2, 1/2, …, 1/2);
 3   for $i$=1 to $N$ do
 4     for $j$=1 to $|I|$ do
 5       Generate $X_{ij}$ with a probability of $p'_{0,j}$;
 6     end for
 7   end for
 8   Sort the $N$ itemsets by descending order of utility, and denote them as
     $S_1, S_2, \dots S_N$;
 9   Initialize the set of top-$k$ HUIs $KH$ with $S_1, S_2, \dots, S_k$;
10   Calculate $\gamma_t$ using Eq. 2;
11   $t$ = 1;
```

In Algorithm 2, we first construct the bitmap representation of the database in Step 1. Step 2 initializes all of the probabilities in the probability vector to 1/2. That is the probability of being one or zero is 0.5. The loop (Steps 3–7) initializes the *N* itemsets of the first iteration. Step 8 reorders the itemsets by descending order of utility. Step 9 initializes *KH* according to the sample of the first iteration. Step 10 then calculates the ρ sample quantile. Finally, Step 11 sets the iteration number to one.

5 Performance Evaluation

In this section, we evaluate the performance of our TKU-CE algorithm and compare it with the TKU [13] and TKO [11] algorithms. We downloaded the source code of the two comparison algorithms from the SPMF data mining library [4].

5.1 Test Environment and Datasets

We conducted the experiments on a computer with a 4-Core 3.40 GHz CPU and 8 GB memory running 64-bit Microsoft Windows 10. We wrote our programs in Java. We used both synthetic and real datasets to evaluate the performance of the algorithms. We generated two synthetic datasets from the IBM data generator [1]: T25I100D5k and T35I100D7k. The parameters are as follows: T is the average size of the transactions, I is the total number of different items, and D is the total number of transactions. We downloaded real datasets from the SPMF data mining library [4]. Both Chess and Connect datasets originate from game steps. Table 3 gives the characteristics of the datasets used in the experiments.

Table 3. Characteristics of datasets used for experiment evaluations

Datasets	Avg. trans. length	No. of items	No. of trans
T25I100D5k	25	100	5,000
T35I100D7k	35	100	7,000
Chess	37	76	3,196
Connect	43	130	67,557

The two real datasets already contain the utility information, while the two synthetic datasets do not contain the utility value or quantity of each item in each transaction. As in TKU [13] and TKO [11], we generate the unit profits for items between 1 and 1,000 using a log-normal distribution and generate the item quantities randomly between 1 and 5.

For all experiments, we set the sample number to 2,000, ρ to 0.2, and the maximum number of iterations to 2,000.

5.2 Runtime

We demonstrate the efficiency of our algorithm and the comparison algorithms with a varying number of desired itemsets, namely *k*, for each dataset.

Figure 1(a) shows the execution time comparison between the algorithms on the T25I100D5k dataset. As *k* increases from 20 to 100, TKU-CE is 8.70 times faster than TKU, and one order of magnitude faster than TKO, on average. In this set of experiments, TKU is faster than TKO. This is because the utility list and the pre-evaluation matrix structure of TKO are not effective on this sparse dataset, and these operations add computational complexity with respect to the other two algorithms.

Figure 1(b) shows the execution time comparison between the algorithms on the T35I100D7k dataset as *k* increases from 3 to 11. For this dataset, TKU did not return any results even after four days, thus, we did not plot its results. TKU-CE's superior efficiency over TKO is more obvious on T35I100D7k. TKU-CE is consistently three orders of magnitude faster than TKO. For 11 itemsets, the runtime comparison between TKU-CE and TKO is 35.696 seconds to 223,016.616 seconds, respectively. Because TKO incurs a very long runtime for this dataset, we set a low number of desired itemsets to obtain the output within a reasonable time.

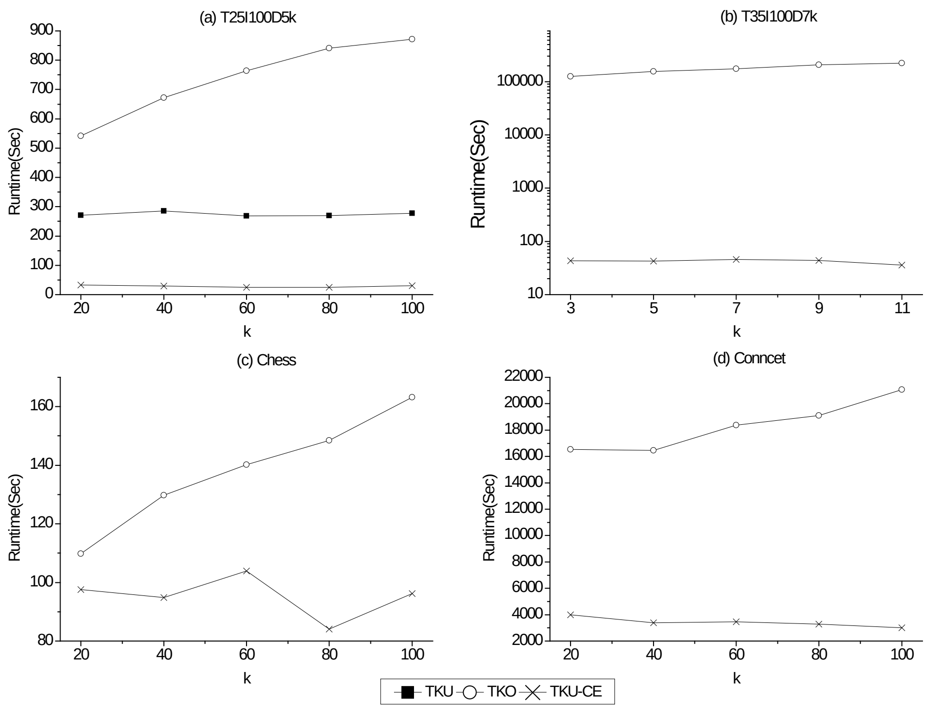

Fig. 1. Execution times for the four datasets

For the Chess dataset, when the range of *k* is between 20 and 100, TKU runs out of memory. Thus, we again omit the results of TKU in Fig.1(c). This time, TKU-CE is faster than TKO by 46.09% on average.

As with the results on Chess, we did not plot the TKU results in Fig.1(d) because it again runs out of memory for the Connect dataset. On average, TKU-CE is 4.43 times faster than TKO.

As we can see from Fig.1, the proposed heuristic algorithm TKU-CE always demonstrates superior efficiency. With respect to the other two algorithms, the one-phase TKO algorithm outperforms the two-phase TKU algorithm in most cases. The runtime of TKU-CE does not increase as *k* increases because when the sample number *N* is greater than *k*, *k* only affects the initialization and update of the set of top-*k* HUIs. This renders TKU-CE not only efficient but also easy to implement.

5.3 Memory Consumption

We also compare the memory usage of the three algorithms for the four datasets. The results are shown in Fig. 2. For the same reasons stated in Sect. 5.2, we only plot the results of TKU for T25I100D5k.

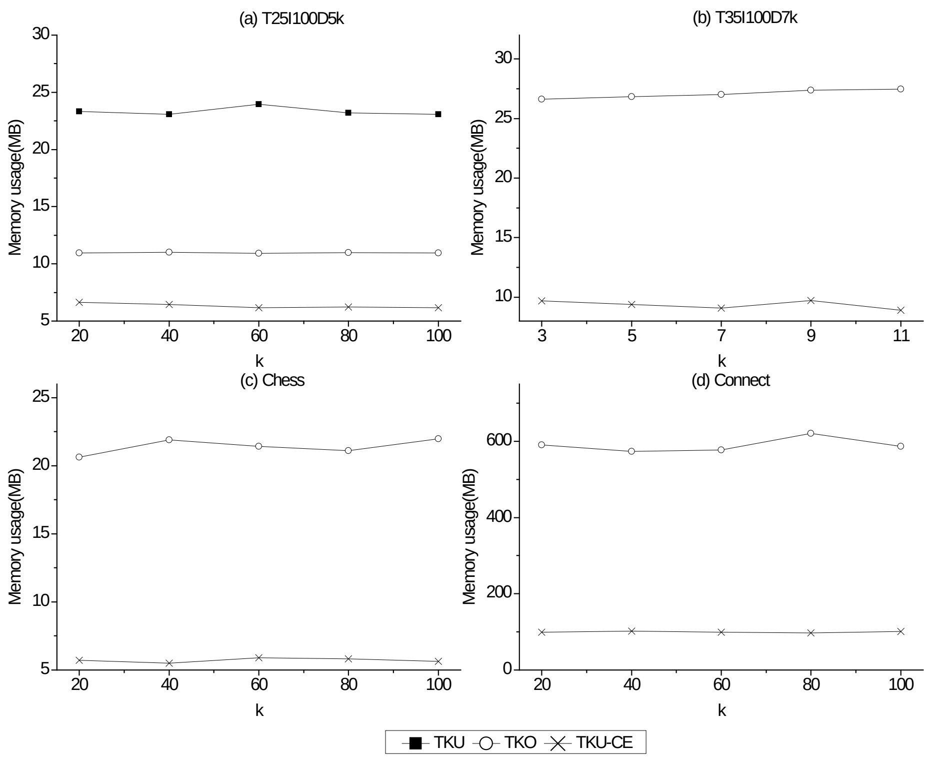

Fig. 2. Memory usage for the four datasets

Figure 2(a) shows the memory usage for the T25I100D5k dataset. On average TKU-CE consumes 2.69 times less memory than TKU, and 73.45% less memory than TKO. Although Fig.1(a) shows that TKU is more efficient than TKO, TKO outperforms TKU in terms of storage consumption.

Figures 2(b) and 1(b) show that the proposed TKU-CE algorithm is not only remarkably more efficient than TKO, but also uses very little memory—less than 10MB on T35I100D7k.

Figures 2(c) and 2(d) show the memory consumption for the two real datasets Chess and Connect. TKU-CE still outperforms TKO with respect to memory usage. On average TKU-CE consumes 2.75 times and 4.92 times less memory than TKO on Chess and Connect, respectively.

From Fig. 2, we can see that TKU-CE consumes less memory than TKU and TKO on these four datasets. Furthermore, the memory usage of TKU-CE is nearly constant. This is because TKU-CE does not consider additional tree or list structures for storing and transforming the original information and does not include any threshold raising and pruning strategies. Thus, it is more suitable for the problem of top-*k* HUI mining without a user-specified minimum utility threshold.

5.4 Accuracy

A heuristic top-*k* HUI mining algorithm cannot ensure the discovery of all of the correct itemsets within a certain number of cycles; that is, some itemsets discovered by TKU-CE may not correspond to the actual top-*k* itemsets of the entire dataset. In this section, we compared the percentage of discovered actual top-*k* HUIs by *k*. We use the TKO algorithm to discover the actual complete list of top-*k* HUIs from the four datasets. We use the following equation to calculate the accuracy of top-*k* HUIs discovered by TKU-CE.

$$Acc_k = CE_k / k \times 100\% \tag{7}$$

where CE_k is the number of actual top-*k* HUIs discovered by TKU-CE.

Table 4. Accuracy for the four datasets

T25I100D5k	k	20	40	60	80	100
	Acc_k (%)	100	100	90	91.25	75
T35I100D7k	k	3	5	7	9	11
	Acc_k (%)	100	100	100	100	100
Chess	k	20	40	60	80	100
	Acc_k (%)	100	100	100	97.50	99
Connect	k	20	40	60	80	100
	Acc_k (%)	95	100	100	96.25	97

Table 4 shows that TKU-CE can discover more than 90% of the actual top-*k* HUIs within 2000 iterations, except when *k* is set to 100 for the T25I100D5k dataset. Furthermore, TKU-CE outputs 100% of true top-*k* HUIs with 12 times for the four datasets. As the number of experiments for each dataset is 5, the probability of TKU-CE returning the exact top-*k* HUIs is 60%. Thus, we see that TKU-CE outputs most of the actual top-*k* HUIs within less time and consuming less memory.

6 Conclusions

In this paper, we heuristically tackle top-*k* HUI mining by proposing a novel CE-based algorithm called TKU-CE. In contrast to existing one-phase or two-phase algorithms, TKU-CE approaches the optimal results using a stochastic approach. The TKU-CE algorithm does not use additional tree or list structures to represent or transform the original information. Furthermore, we also avoid the widespread threshold raising and pruning strategies of existing related algorithms. Experimental results on both synthetic and real datasets show that TKU-CE can discover a suitable number of top-*k* HUIs with high efficiency and low memory usage.

Acknowledgments. This work was partially supported by the National Natural Science Foundation of China (61977001), the Great Wall Scholar Program (CIT&TCD20190305), and Beijing Urban Governance Research Center.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499 (1994)
2. de Boer, P.-T., Kroese, D. P., Mannor, S., Rubinstein, R. Y.: A tutorial on the cross-entropy method. Annals OR. **134**(1), 19-67 (2005)
3. Duong, Q.-H., Liao, B., Fournier-Viger, P., Dam, T.-L.: An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. Knowl.-Based Syst. **104**, 106-122 (2016)
4. Fournier-Viger, P., Lin, C. W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T.: The SPMF open-source data mining library version 2. In: Berendt, B., Bringmann, B., Fromont, É., Garriga, G., Miettinen, P., Tatti, N., Tresp, V. (eds.) ECML PKDD 2016. LNCS, vol. 9853, pp. 36-40. Springer, Cham (2016)
5. Kannimuthu, S., Premalatha, K.: Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl. Artif. Intell. **28**(4), 337-359 (2014)
6. Lin, J. C.-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 (2017)
7. Liu, Y., Liao, W.-K., Choudhary, A. N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T. B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS, vol.3518, pp.689-695. Springer, Heidelberg (2005)
8. Ryang, H., Yun, U.: Top-k high utility pattern mining with effective threshold raising strategies. Knowl.-Based Syst. **76**, 109-126 (2015)
9. Singh, K., Singh, S. S., Kumar, A., Biswas, B.: TKEH: an efficient algorithm for mining top-k high utility itemsets. Appl. Intell. **49**(3),1078–1097 (2019)
10. Song, W., Huang, C.: Discovering high utility itemsets based on the artificial bee colony algorithm. In: Phung, D., Tseng, V., Webb, G., Ho, B., Ganji, M., Rashidi, L. (eds) PAKDD 2018. LNCS, vol. 10939, pp. 3-14. Springer, Cham (2018)
11. Tseng, V. S., Wu, C.-W., Fournier-Viger, P., Yu, P. S.: Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. **28**(1), 54-67 (2016)
12. Wang, J., Han, J., Lu, Y., Tzvetkov, P.: TFP: An efficient algorithm for mining top-k frequent closed itemsets. IEEE Trans. Knowl. Data Eng. **17**(5), 652-664 (2005)
13. Wu, C.-W., Shie, B.-E., Tseng, V. S., Yu, P. S.: Mining top-k high utility itemsets. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.78-86 (2012)
14. Wu, J. M. T., Zhan, J., Lin, J. C. W.: An ACO-based approach to mine high-utility itemsets. Knowl.-Based Syst. **116**, 102–113 (2017)