Applied Intelligence
https://doi.org/10.1007/s10489-021-02576-z

Heuristically mining the top-*k* high-utility itemsets with cross-entropy optimization

Wei Song[1] · Chuanlong Zheng[1] · Chaomin Huang[1] · Lu Liu

Accepted: 28 May 2021

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* HUIs without setting a utility threshold is becoming an alternative to determine all the HUIs. In this paper, we propose two algorithms, called the top-*k* high-utility itemset mining based on cross-entropy method (TKU-CE) and TKU-CE+, for mining the top-*k* HUIs heuristically. The TKU-CE algorithm is based on cross-entropy, and implements 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. TKU-CE+ optimizes TKU-CE in three respects. First, unpromising items are filtered by critical utility value, to reduce the computational burden in the initial stage. Second, a sample refinement strategy is used in each iteration, to reduce the computational burden in the iterative stage. Finally, smoothing mutation is proposed, to randomly generate some new itemsets in addition to those from previous iterations. Consequently, diversity of samples is improved, so that more actual top-*k* HUIs can be discovered with fewer iterations. Compared with state-of-the-art algorithms, TKU-CE and TKU-CE+ are easy to implement and avoid the computational costs that would be incurred by additional data structures and threshold-raising strategies. Extensive experimental results show that both algorithms are efficient, memory-saving, scalable, and can discover the most actual top-*k* HUIs.

Keywords Top-k high-utility itemset · Cross-entropy · Critical utility value · Sample refinement · Smoothing mutation · Bit edit distance

1 Introduction

The discovery of sets of items in transaction databases that occur with high frequency, known as frequent itemset mining (FIM) [22], is a major topic in data mining and has many applications [8]. However, FIM has two main drawbacks. First, the frequency metric in FIM does not reflect the profit or utility of different items. Thus, items with low frequency but high profit may be ignored. To solve this problem, high-utility itemset mining (HUIM) [21, 31], which extends FIM, is used to discover high-profit itemsets by considering both the quantity and value of a single item. Various algorithms for mining typical high-utility itemsets (HUIs) [4, 5], other useful variations of HUIs [10, 14, 23, 32], and HUIM algorithms under a new computing framework [35, 37], have been developed. Second, a minimum support threshold is required to identify the qualifying itemsets. However, it is difficult for non-expert users to set an appropriate threshold. Consequently, top-*k* FIM has attracted the attention of researchers [3, 34] because it considers the *k* most-frequent itemsets as the mining target. The *k* value is a more intuitive and direct parameter for users to set than the minimum threshold.

Combining both HUIM and top-*k* FIM, top-*k* HUIM [30, 36] has emerged as a solution to the drawbacks of FIM

This article is part of the Topical Collection on Emerging topics in Applied Intelligence selected from IEA/AIE2020

✉ Wei Song
songwei@ncut.edu.cn

Chuanlong Zheng
zhengcl1234@163.com

Chaomin Huang
hcm6233@gmail.com

Lu Liu
18810377067@163.com

[1] School of Information Science and Technology, North China University of Technology, Beijing, China

Published online: 29 July 2021

outlined above. Top-k HUIM uses the same concept of utility as HUIM; that is, an item's utility is mainly reflected by the product of its profit and occurrence frequency. As in the mining of the k most-frequent itemsets, the k itemsets with the highest utility values are the target of top-k HUIM. Wu et al. [36] introduced the top-k HUIM problem, and proposed the TKU algorithm for mining the top-k HUIs. TKU first determines candidates and then filters the actual top-k HUIs; this implies that TKU is a two-phase algorithm. Two-phase methods often generate too many candidate itemsets before identifying the actual top-k HUIs. The candidate generation process is quite expensive, particularly on transactional databases with many transactions and distinct items. Therefore, the TKO algorithm, which uses a one-phase procedure, was proposed for mining top-k HUIs [33]. There are several other one-phase top-k HUIM algorithms, such as TKUL-Miner [17] and kHMC [7], that have also been proposed. In contrast to two-phase algorithms, one-phase algorithms discover all the top-k HUIs directly, without an expensive candidate generation process, by using structures such as utility lists or set enumeration trees.

Regardless of whether top-k HUIM is performed using a two-phase or one-phase algorithm, it is equivalent to HUIM with the minimum utility threshold set to zero. Therefore, 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. In this paper, we aim to use a different strategy that does not require constant threshold raising. We achieve this goal by using the cross-entropy (CE) method.

The CE method is a combinatorial and multi-extremal optimization technique [2] that is applied in fields such as vehicle classification [18], reinforcement learning [12], and neural networks [1]. Like other heuristic techniques, the CE method approaches the optimal values using an iterative procedure, in which 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, good results are retained and poor results are abandoned in each iteration. After many iterations, the best N results are obtained. This method is essentially consistent with the top-k HUIM problem. Therefore, we use the CE method to formulate the novel top-k HUIM algorithms proposed in this paper.

This paper is an extended version of our conference paper [30], in which TKU-CE was proposed. The contributions of the present paper are as follows:

1) We formally model the top-k HUIM problem from the perspective of combinatorial optimization, in which a binary vector is used to represent a possible solution and the utility value is optimized directly.
2) We propose TKU-CE, which is a top-k HUIM algorithm that performs typical CE optimization. In TKU-CE, the top-k HUIs are discovered iteratively with a probability vector (PV) that is initialized randomly. Within each iteration, itemsets of the new sample are generated with the current PV, and then a new PV is updated according to the new sample; this uses the principle that items in itemsets with higher utility values tend to have higher new probabilities. Finally, the top-k HUIs can be identified after all iterations terminate.
3) We propose TKU-CE+, which improves the baseline TKU-CE algorithm in three respects. First, we prove that, except for the k items with the highest utility values, other items do not need to be further considered. This reduces the memory usage. Second, only itemsets with high utility values are retained and used to generate new itemsets in each iteration. Therefore, the search space is reduced gradually in each iteration. Finally, in addition to the itemsets retained from previous iterations, some new itemsets are also generated randomly. Consequently, the diversity of the sample is improved and more accurate top-k HUIs can be discovered in early iterations.
4) Finally, we use both synthetic and real datasets in experimental evaluations. The experimental results show that TKU-CE and TKU-CE+ are efficient and scalable, and require less memory space. Furthermore, they can discover more than 90% of the actual top-k HUIs, in most cases.

The remainder of this paper is organized as follows. We formalize the top-k HUIM problem and the CE method in Section 2. In Section 3, we review related work. We describe the proposed TKU-CE and TKU-CE+ algorithms in detail in Section 4. We present the experimental results on both synthetic and real datasets in Section 5. In Section 6, we conclude this paper.

2 Preliminaries

2.1 Top-k HUIM problem

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items. A set $X \subseteq \boldsymbol{I}$ is called an *itemset*; an itemset that contains k items is called a k-itemset. Let $\boldsymbol{D} = \{T_1, T_2, \ldots, T_n\}$ be a transaction database. Each transaction $T_d \in \boldsymbol{D}$, with the unique identifier d, 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 (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$.

The *minimum utility threshold* δ, specified by the user, is defined as a percentage of the total TU values of the database, whereas the *minimum average-utility value* is defined as $min\_util = \delta \times \sum_{T_d \in D} TU(T_d)$. An itemset X is called an *HUI* if $u(X) \geq min\_util$. Given a transaction database $\boldsymbol{D}$, the task of HUIM is to identify all itemsets whose utility is no less than *min_util*. The set of all HUIs in $\boldsymbol{D}$ with respect to *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 fewer than k itemsets whose utilities are greater than $u(X)$ in $f_H(\boldsymbol{D}, 0)$. Top-k HUIM aims to discover the k itemsets with the highest utilities, where k is a parameter set by the user.

As a running example, consider the transaction 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 $u(C, T_1) = 3 \times 1 = 3$, 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 TU of T_5 is $TU(T_5) = u(AC, T_5) = 13$. The utilities of the other transactions are presented 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 CE optimization

Like other heuristic algorithms, CE optimization approaches the optimal values iteratively. 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 [2]. Let $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ be an n-dimensional binary vector; that is, the value of y_j $(1 \leq j \leq n)$ 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:

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

Table 1 Example database

TID	Transactions	TU
1	(A, 1) (B, 1) (C, 1) (F, 2)	19
2	(B, 1) (D, 1) (E, 1)	9
3	(A, 1) (B, 1) (C, 1) (F, 1)	15
4	(C, 3) (D, 2) (F, 1)	17
5	(A, 1) (C, 2)	13

Table 2 Profit table

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

A naive approach 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 $\boldsymbol{y}$, which leads to $S(\boldsymbol{x}) = n$. The elements of the trial binary vector $\boldsymbol{x}$, that is, $x_1, x_2, \ldots, x_n$, are independent Bernoulli random variables with success probabilities $p_1, p_2, \ldots, p_n$, and these probabilities comprise a PV $p' = (p_1, p_2, \ldots, p_n)$. The CE method for COP consists of creating sequences 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 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ of Bernoulli vectors, $S(\boldsymbol{x_i})$ is calculated for all i, and the elements are sorted in descending order of $S(\boldsymbol{x_i})$. Let γ_t be a ρ sample quantile of the performances; that is,

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

where $\lceil \rho \times N \rceil$ is the smallest integer that is greater than or equal to $\rho \times N$. The PV is then updated by

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

where $j = 1, 2, \ldots, n$, $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, 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.

Equation (3) is used iteratively to update the PV until the termination criterion is satisfied. There are two possible termination criteria: γ_t has not changed for a number of consecutive iterations or the PV has converged to a binary vector.

3 Existing algorithms

3.1 Top-*k* HUIM

The basic concepts of top-k HUIM were described by Wu et al. [36]. In contrast to top-k FIM, the utility of an itemset in top-k HUIM is neither monotone nor anti-monotone. Thus, the anti-monotonicity-based pruning strategies of top-k FIM [3, 34] cannot be used directly for top-k HUIM. The first top-k HUIM algorithm, TKU, introduced the concept of the optimal

minimum utility threshold to reduce the high computational complexity incurred using zero as the minimum threshold, and used a threshold-raising method to improve mining efficiency. Using five threshold-raising strategies and four pruning properties, TKU first generates candidate itemsets and then filters the actual top-*k* HUIs, following the two-phase procedure that is widely used in HUIM algorithms [21].

REPT [25], similar to TKU, is another top-*k* HUIM 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 the exact and pre-evaluated utilities of itemsets with a length of one or two, to reduce the number of candidates. However, REPT requires the user to set a parameter *N*, whose selection is non-trivial.

The above two-phase algorithms generate a very large number of candidates, potentially incurring a high execution time and storage cost. Recent algorithms focus on mining top-*k* HUIs directly in a single phase, without generating candidates. The TKO algorithm [33] uses vertical data representation to transform the original database and uses a utility list data structure to maintain itemset information during the mining process. Furthermore, TKO uses three pruning strategies to facilitate the mining process.

kHMC [7] also mines the top-*k* HUIs in one phase. As in TKO, kHMC uses a utility list. Furthermore, kHMC proposes the concept of coverage to raise the intermediate thresholds. As in other related algorithms, kHMC uses four threshold-raising strategies and five pruning properties to improve mining speed.

THUI [16] is another algorithm that mines the top-*k* HUIs in one phase. THUI uses a leaf itemset utility (LIU) matrix structure to store the utilities of itemsets. Two threshold-raising strategies, that is, LIU exact and LIU lower bound estimation, are used to accelerate mining performance.

For existing top-*k* HUIM algorithms, the major challenge that differentiates top-*k* HUIM from traditional HUIM is the use of threshold-raising strategies, for both two-phase and one-phase algorithms; that is, gradually increasing the minimum utility threshold constricts the search space during the mining process. Thus, new methods that achieve satisfactory performance without using threshold-raising strategies are promising for top-*k* HUIM.

3.2 HUIM using heuristic methods

Inspired by biological and physical phenomena, heuristic methods are effective for solving combinatorial problems. Based on stochastic methods, heuristic methods can explore very large search spaces to find near-optimal solutions. Heuristic methods have been used to traverse immense candidate itemset spaces within an acceptable time for FIM [6] and HUIM [27].

Two HUIM algorithms based on the genetic algorithm (GA) were proposed in [13]. The difference between them is whether a minimum utility threshold is required. Premature convergence is the main problem of these two algorithms; that is, the algorithms easily fall into local optima. Zhang et al. also proposed an algorithm for mining HUIs based on GA, called HUIM-IGA [39]. For HUIM-IGA, four strategies—neighborhood exploration, population diversity improvement, invalid combination avoidance, and HUI loss prevention—are used to improve the algorithm's performance.

Particle swarm optimization (PSO) is another heuristic method used for mining HUIs. HUIM-BPSO$_{sig}$ [20] and HUIM-BPSO [19] are two PSO-based algorithms for mining HUIs. HUIM-BPSO outperforms HUIM-BPSO$_{sig}$ using an OR/NOR-tree structure. HUIM-BPSO-nomut [11] is another PSO-based algorithm for mining HUIs. In contrast to HUIM-BPSO$_{sig}$ and HUIM-BPSO, no utility threshold is required for HUIM-BPSO-nomut.

Wu et al. proposed an algorithm based on ant colony optimization, called HUIM-ACS, for mining HUIs [38]. This algorithm generates a routing graph before all the ants start their tours. An ant might generate several candidate itemsets during a tour. Therefore, each node in the routing graph represents a specific itemset that can be evaluated, to determine whether it is an HUI. Using this graph both avoids the recalculation of the utility value for an itemset and enables the algorithm to check whether all the itemsets have been considered. Furthermore, this strategy uses positive pruning and recursive pruning to improve the algorithm's efficiency.

We also studied the HUIM problem from the perspective of the artificial bee colony (ABC) algorithm. The proposed HUIM-ABC discovers HUIs by modeling the itemsets as nectar sources [26]. For each nectar source, three types of bee—employed bee, onlooker bee, and scout bee—are used for sequential optimization within an iteration. During one iteration, the algorithm outputs an itemset when it is verified as an HUI. This process is executed iteratively until the maximal number of cycles is reached. In HUIM-ABC, the resulting itemsets' lengths are used to generate HUIs as early as possible.

There are also HUIM algorithms that are based on other heuristic algorithms, such as wolf optimization [24] and binary differential evolution [15].

In contrast to the above methods, a general framework called Bio-HUIF was proposed [27]. In Bio-HUIF, all discovered HUIs, rather than those with a high utility, are proportionally maintained in the next generation. Thus, the population is diverse, which is more suitable for the HUIM problem. Furthermore, Bio-HUIF is also used for discovering high-average-utility itemsets [28].

In this paper, we solve the top-*k* HUIM problem using CE as the central approach. Our approach differs from previous related heuristic algorithms in two respects. First, most

existing heuristic algorithms are intended to discover all HUIs, whereas the aim of our algorithms is to discover top-*k* HUIs. Second, most previous heuristic algorithms are based on evolutionary algorithms, whereas CE, the basis of our algorithms, is intended for estimating probabilities of rare events in complex stochastic networks. Moreover, TKU-CE and TKU-CE+ differ from related exact top-*k* HUIM algorithms in that no additional data structures or threshold-raising strategies are required.

4 Proposed algorithms

In this section, we describe the two algorithms: TKU-CE and its improved version, TKU-CE+. We first introduce the bitmap data representation structure and then model the top-*k* HUIM problem using the CE method. Finally, we explain the two proposed algorithms in detail.

4.1 Bitmap item information representation

Understanding the method used to transform the original database is essential for understanding the two proposed algorithms. We use a bitmap, which is an effective representation for item information in FIM and HUIM algorithms, in TKU-CE and TKU-CE+ to identify transactions that contain the target itemsets. This representation allows the utility values of the target itemsets to be calculated efficiently using bitwise operations.

Specifically, a *bitmap cover* is used for representing 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 extends naturally to itemsets. Let *X* and *Y* be two itemsets. *Bit*(*X*) corresponds to the bitmap cover that represents the transaction set for *X*. *Bit*(*X*∪*Y*) can be computed as *Bit*(*X*)∩*Bit*(*Y*), that is, the bitwise-AND of *Bit*(*X*) and *Bit*(*Y*).

Table 3 shows the bitmap representation of the example database in Table 1. For example, item B appears in transactions T_1, T_2, and T_3. Hence, the bitmap cover of B, that is, *Bit*(B), is <11100>.

Table 3 Bitmap representation of the example database

	A	B	C	D	E	F
1	1	1	1	0	0	1
2	0	1	0	1	1	0
3	1	1	1	0	0	1
4	0	0	1	1	0	1
5	1	0	1	0	0	0

4.2 Modeling top-*k* HUIM based on CE optimization

After transforming the database to a bitmap, it is natural to encode each solution (itemset) in a binary vector. In this paper, a binary vector representing an itemset is called an *itemset vector* (IV).

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 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ in descending order of $S(\boldsymbol{x}_i)$ $(1 \le i \le N)$, and update the sample quantile γ_t and PV p'_t accordingly.

4.3 Proposed mining algorithm: TKU-CE

Algorithm 1 describes the top-*k* HUIM algorithm TKU-CE.

Algorithm 1	TKU-CE
Input:	Transaction database ***D***, number of desired HUIs *k*, number of samples *N*, quantile parameter ρ, maximum number of iterations *max_iter*
Output:	Top-*k* HUIs
1	$p'_0 = (1/2, 1/2, \ldots, 1/2)$;
2	$t = 1$;
3	Initialization(***D***, *k*, *N*, ρ);
4	**while** $t \le max\_iter$ **and** p'_t is not a binary vector **do**
5	Calculate p'_t using Eq. (3);
6	**for** *i*=1 to *N* **do**
7	**for** *j*=1 to |*I*| **do**
8	Generate X_{ij} with the probability of $p'_{t,j}$;
9	**end for**
10	**end for**
11	Sort the *N* itemsets in utility-descending order and denote the results by $S_1, S_2, \ldots, S_N$;
12	Update the set of top-*k* HUIs *KH* using the new sample;
13	Calculate γ_t using Eq. (2);
14	*t*++;
15	**end while**
16	Output top-*k* HUIs.

In Algorithm 1, Step 1 first initializes all the probabilities in the PV to 1/2; that is, the probability of each bit being one or zero is 0.5. Then, Step 2 sets the iteration number to one. Next, the procedure Initialization, described in Algorithm 2, is called in Step 3. The main loop (Steps 4–15) calculates the top-*k* HUIs iteratively. In addition to the maximal number of iterations, the PV becoming a binary vector is also a termination criterion. With a binary PV, all the *N* itemsets are the same in one iteration because each item in each itemset is definitely one or zero. Consider the following running example. There are six items: A, B, C, D, E, and F. After several iterations, the PV converges to (1, 1, 1, 0, 0, 1), and then the itemsets within the sample are all ABCF because the probabilities of the bits corresponding to D and E are zero. Step 5 calculates the PV of

the new iteration. The loop (Steps 6–10) generates N new itemsets in a bitwise manner. Here, $|\boldsymbol{I}|$ is the number of items in $\boldsymbol{I}$. Specifically, for itemset X_i and its jth bit, we randomly generate a probability $p_{i,j}$, and then determine the value X_{ij}:

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

Step 11 arranges the itemsets in descending order of utility and renames them $S_1, S_2, \ldots, S_N$. Step 12 updates KH, that is, the set of top-k HUIs, according to the latest sample. Step 13 calculates the new ρ sample quantile. Step 14 increments the iteration number by one. Finally, Step 16 outputs all the discovered top-k HUIs.

It should be noted that the number of resulting itemsets of top-k HUIM may not exactly equal k. As in TKU [36] and TKO [33], we output the actual results regardless of whether the number of results is k.

Algorithm 2	Initialization($\boldsymbol{D}$, k, N, ρ)
1	Represent the database using a bitmap;
2	**for** i=1 to N **do**
3	**for** j=1 to $\|\boldsymbol{I}\|$ **do**
4	Generate X_{ij} with a probability of $p'_{0,j}$;
5	**end for**
6	**end for**
7	Sort the N itemsets in utility-descending order and denote them as $S_1, S_2, \ldots, S_N$;
8	Initialize the set of top-k HUIs KH with $S_1, S_2, \ldots, S_k$;
9	Calculate γ_i using Eq. (2);

In Algorithm 2, Step 1 first constructs the bitmap representation of the database. As in Algorithm 1, the loop (Steps 2–6) initializes the N itemsets of the first iteration. Step 7 sorts the itemsets in descending order of utility. Step 8 initializes KH according to the sample in the first iteration. Finally, Step 9 calculates the ρ sample quantile.

Most steps, except Step 1, of Algorithm 2 are similar to steps in the main loop (Steps 4–15) of Algorithm 1. As the names of the two algorithms suggest, the most important difference between Algorithms 1 and 2 is that the latter is the initialization phase of the former. That is, Algorithm 2 performs the initial actions of the main loop of Algorithm 1. With the above initialization, the steps in the main loop of Algorithm 1 are calculated and updated iteratively.

The complexity of TKU-CE depends on the maximum number of iterations max_iter, the number of transactions N, and the number of items $|\boldsymbol{I}|$ in a database. Because the relationship between these three variables is a triple loop, the computational complexity of TKU-CE in the worst case is $O(max\_iter \times N \times |\boldsymbol{I}|)$.

4.4 Improved mining algorithm: TKU-CE+

TKU-CE models the top-k HUIM problem directly using CE optimization, and its performance can be further improved from the following three observations.

First, all items are considered in the initial stage of TKU-CE, and the length of the binary vectors is equal to the number of single items. If we ignore unpromising items in the initial stage and reduce the length of the IVs, the search space reduces and the mining process accelerates.

Second, because the number of top-k HUIs is often significantly less than the total number of HUIs, it is possible that not all IVs contribute to the final results equally. If we only consider the most important IVs and ignore the unpromising IVs, the mining performance can be improved during the iterative stage.

Third, heuristic-based pattern mining algorithms cannot ensure the discovery of all the correct results within a certain number of iterations [6, 19]. The locally optimal solution is one of the most typical problems encountered by this type of algorithm. Because increasing the population diversity is an effective solution [27], improving the diversity of each sample may also lead to high-quality results, in CE-based top-k HUIM.

4.4.1 Critical utility value

In this subsection, we propose the critical utility value (CUV) for pruning unpromising itemsets in the initial stage.

Definition 1 Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items in database $\boldsymbol{D}$ and $util_1, util_2, \ldots, util_m$ be the utility of the m single items, such that, for any $1 \le p < q \le m$, $util_p \ge util_q$; we call $util_k$ the CUV of $\boldsymbol{D}$, denoted by $CUV(\boldsymbol{D})$.

From Definition 1, the CUV is the k-th highest utility of the 1-itemset. For top-k HUIM, the CUV can be used as a pruning strategy with the concept of transaction-weighted utilization (TWU). As defined in [21], the TWU of itemset X is the sum of the transaction utilities of all the transactions containing X, that is, $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$.

Theorem 1 Let $\boldsymbol{D}$ be a database and i_x be a 1-itemset in $\boldsymbol{D}$. If $TWU(i_x) < CUV(\boldsymbol{D})$, i_x and all itemsets containing i_x are not top-k HUIs.

Proof Let $umin$ be the minimal utility of top-k HUIs in $\boldsymbol{D}$. According to the definition of top-k HUIs, $umin$ is also the k-th highest utility of all itemsets in $\boldsymbol{D}$. Because $CUV(\boldsymbol{D})$ is the k-th highest utility of 1-itemsets, $CUV(\boldsymbol{D}) \le umin$.

Because $u(i_x) \le TWU(i_x)$ and $TWU(i_x) < CUV(\boldsymbol{D})$, $u(i_x) < CUV(\boldsymbol{D})$ holds. Hence, $u(i_x) < umin$, which implies that i_x is not a top-k HUI.

Consider an arbitrary itemset X containing i_x, $u(X) \le TWU(i_x) < CUV(\boldsymbol{D}) \le umin$. Thus, X is not a top-k HUI.

Using Theorem 1, once a 1-itemset is found to have a TWU that is less than $CUV(\boldsymbol{D})$, this itemset and all its supersets can be pruned safely. We call this strategy CUV-based pruning.

In the running example, $TWU(\mathrm{E}) = TU(T_2) = 9$. The TWU and utility of the six single items are listed in Table 4. If we want to discover the top-3 HUIs, then $CUV(\boldsymbol{D}) = u(\mathrm{F}) = 16$. Because $TWU(\mathrm{E}) < CUV(\boldsymbol{D})$, E and all its supersets can be ignored in the subsequent mining process. Thus, the length of IVs in TKU-CE is six, whereas the length of IVs in TKU-CE+ is five.

4.4.2 Sample refinement

For typical itemset mining algorithms based on heuristic methods, the sample size always remains stable. However, in contrast to typical HUIM, only the k itemsets with the greatest utility are required for top-k HUIM, rather than all itemsets whose utility is no less than the minimum utility threshold. Compared with typical HUIM, the number of results of top-k HUIM is small. Thus, the following questions arise. As the number of iterations increases, do all IVs play the same role in generating top-k HUIs? Can the number of IVs be gradually reduced to improve mining performance?

We study this problem from the perspective of IVs whose utility values are no less than the ρ sample quantile, namely γ_t.

Definition 2 Let V_X be an IV representing itemset X. If $u(X) \geq \gamma_t$, we call V_X an *elite IV*. We call a sample that is only composed of elite IVs an *elite sample.*

From Definition 2, it follows that the size of an elite sample is no greater than the size of a typical sample used in TKU-CE. Furthermore, as the number of iterations increases, the size of an elite sample will gradually decrease. Therefore, the search space will be reduced accordingly, and the computational cost will be decreased. We call this strategy the *sample refinement strategy*.

4.4.3 Smoothing mutation

In this subsection, we propose the smoothing mutation, to increase the diversity of the IVs in each iteration.

Mutation is a classical concept in GA that is used to avoid locally optimal solutions. With mutation, a more diverse population can be generated, which increases the chance of finding a globally optimal solution. Inspired by the idea of mutation, in addition to IVs following PVs, TKU-CE+ also generates part of the new IVs randomly in each iteration. This is different from TKU-CE, in which all IVs are generated strictly with the PVs.

Definition 3 Suppose that a sample $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ is sorted in descending order of $u(\boldsymbol{x}_i)$; that is, for any $1 \leq i \leq j \leq N$, $u(\boldsymbol{x}_i) \geq u(\boldsymbol{x}_j)$. The *smooth factor* (SF) α is defined as

$$\alpha = \frac{u(x_1) - u\left(x_{\lceil \rho \times N \rceil}\right)}{u(x_1)} \times \rho, \tag{7}$$

where ρ is the parameter used to generate the sample quantile in Eq. (2).

Using the SF, TKU-CE+ exploits the following strategy to generate the new sample. For each sample, $\lfloor N \times \alpha \rfloor$ IVs are generated randomly and $N - \lfloor N \times \alpha \rfloor$ IVs are generated with the current PV, where $\lfloor N \times \alpha \rfloor$ denotes the greatest integer that is less than or equal to $N \times \alpha$. We call this strategy the *smoothing mutation*.

Consider the running example. We set $N = 100$ and $\rho = 0.2$. Thus, $N \times \rho = 20$. Suppose that, after several iterations, $u(\boldsymbol{x}_1) = 34$ and $u(\boldsymbol{x}_{20}) = 21$, so that $\lfloor N \times \alpha \rfloor = \lfloor 100 \times ((34 - 21)/34) * 0.2 \rfloor = 7$. Consequently, using the smoothing mutation, seven IVs are generated randomly, whereas the other IVs of the next sample are generated following the current PV.

As the number of iterations increases, the PV tends to converge, and the value of $(u(x_1) - u(x_{\lceil \rho \times N \rceil}))$ also decreases. Thus, as the number of iterations increases, fewer IVs are generated from the smoothing mutation.

4.4.4 Algorithm description

Based on the above discussion, TKU-CE+ is described in Algorithm 3. To minimize repetition, we focus on the parts that are different from the original TKU-CE.

Algorithm 3	TKU-CE+
Input:	Transaction database $\boldsymbol{D}$, number of desired HUIs k, number of samples N, quantile parameter ρ, maximum number of iterations *max_iter*
Output:	Top-k HUIs
1	Scan $\boldsymbol{D}$ once to calculate $CUV(\boldsymbol{D})$ and TWUs of all items;
2	Delete unpromising items using CUV-based pruning;
3	Initialize the first PV with each element equal to 1/2;
4	Calculate the first sample of N itemsets and γ_t, and set $t = 1$;
5	**while** $t \leq max\_iter$ **and** p'_t is not a binary vector **do**
6	Calculate p'_t using Eq. (3);
7	Perform sample refinement strategy, and denote the current sample size as Nt;
8	Generate the new sample using the smoothing mutation strategy;
9	Sort the Nt itemsets in utility-descending order and denote the results by $S_1, S_2, \ldots, S_{Nt}$;
10	Update the set of top-k HUIs KH using the new sample;
11	Calculate γ_t using Eq. (2);
12	t++;
13	**end while**
14	Output top-k HUIs.

Algorithm 3 first processes all items to calculate their utility values and TWUs, and determines $CUV(\boldsymbol{D})$ in Step 1. Step 2 then deletes items whose TWU is less than $CUV(\boldsymbol{D})$. Steps 3 and 4 initialize the first PV, generate the first sample, and set the initial iteration number to one. Similarly to the main loop in Algorithm 1, the loop (Steps 5–13) updates the PV, new sample, top-k HUIs, sample quantile, and iteration number, in

Table 4 Utility and TWU of the single items

Item	Utility	TWU
A	21	47
B	3	43
C	21	64
D	6	26
E	6	9
F	16	51

turn. When the loop terminates, the top-k HUIs are output in Step 14.

In summary, TKU-CE+ improves TKU-CE in three respects.

First, the CUV is used in TKU-CE+ for pruning unpromising itemsets in the initial stage. With the CUV, the length of the IVs in TKU-CE+ could be less than that in TKU-CE. Specifically, the length of the IVs is equal to the total number of items in TKU-CE, whereas some unpromising items are not included in the IVs in TKU-CE+. Accordingly, the length of the PVs also decreases to the same length as that of the IVs.

Second, a sample refinement strategy is added for optimization during the iterative stage. This strategy only considers elite IVs in each iteration. Thus, the sample size gradually decreases as the number of iterations increases, and the computational cost of each iteration is reduced accordingly.

Third, smoothing mutation is used to generate more diverse IVs. That is, most of the new IVs are generated by the PVs, whereas a few of the new IVs are generated randomly.

Similarly to Section 4.3, the computational complexity of TKU-CE+ in the worst case is $O(max\_iter \times N \times Len)$, where Len is the length of the IVs in TKU-CE+. The difference between TKU-CE+ and TKU-CE is that Len is no greater than the total number of items, as a consequence of CUV-based pruning.

5 Experimental results

In this section, we evaluate the performance of our algorithms and compare them with the TKU [36], TKO [33], and kHMC [7] algorithms. The source code of TKU and TKO was downloaded from the SPMF data mining library [9], and the source code of kHMC was provided by the author.

5.1 Experimental environment and datasets

The experiments were performed on a computer with a 4-core 3.40-GHz CPU and 8 GB of memory running 64-bit Microsoft Windows 10. Our programs were written in Java. Six datasets were used to evaluate the performance of the algorithms. The characteristics of the datasets are presented in Table 5.

Table 5 Characteristics of the datasets

Dataset	Avg.Trans.Len	#Items	#Trans
Chess	37	75	3196
Chainstore	7.23	46,086	1,112,949
Pumsb	74	2113	49,046
T35I100D7k	35	100	7000
T50I150D10k	50	150	10,000
T40I100D20k	40	100	20,000

We used three real datasets downloaded from the SPMF data mining library [9]. The Chess dataset originates from game steps. The Chainstore dataset is composed of customer transactions from a major grocery store chain in the USA. Pumsb is a dataset of census data for population and housing. These three datasets contain utility information. The other three datasets are synthetic datasets that do not contain a utility value or quantity of each item in each transaction. Using a random transaction database generator provided in SPMF [9], we generated a random unit profit for items, following a Gaussian distribution.

For all experiments, the sample number N was set to 2000, ρ to 0.2, and the maximum number of iterations to 2000.

5.2 Execution time

We assessed the execution time of our algorithms and the comparison algorithms using a varying number of desired itemsets (that is, k) for each dataset.

Figure 1(a) shows a comparison between the execution times of the algorithms on the Chess dataset. As k increased from 20 to 100, both TKU-CE and TKU-CE+ were slightly slower than kHMC, but were faster than TKO. We did not plot the performance of TKU because it did not return any results after 24 h.

On the extremely sparse dataset Chainstore, both TKU and TKU-CE had insufficient memory. Therefore, we omitted their results from Fig. 1(b). As shown in Table 5, there are 46,086 distinct items in Chainstore, and these items are all considered in the IVs of TKU-CE, leading to memory overflow. As a consequence of CUV-based pruning, the IVs of TKU-CE+ omit unpromising items in the initial stage, which avoids the memory problem suffered by TKU-CE. Generally, TKU-CE+ was one order of magnitude faster than TKO, but was slightly slower than kHMC.

On the other real dataset, Pumsb, neither TKU nor TKO returned any results after 24 h. Therefore, their results are not plotted in Fig. 1(c). For the three comparison algorithms, the execution time of kHMC was between those of the two heuristic algorithms: kHMC was faster than TKU-CE but slower than TKU-CE+.

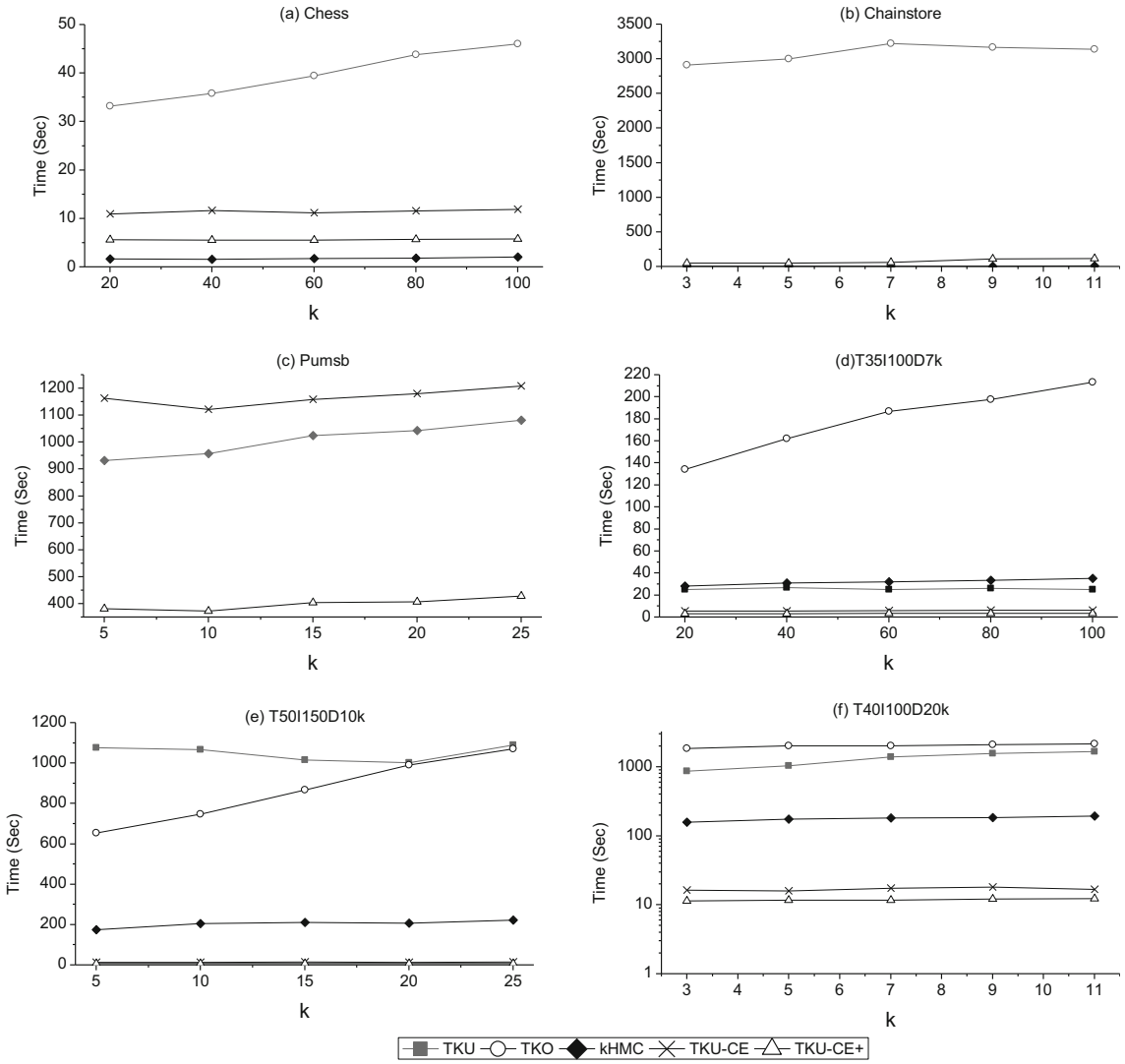


Fig. 1 Execution times for the six datasets

For the three synthetic datasets (T35I100D7k, T50I150D10k, and T40I100D20k), both TKU-CE and TKU-CE+ were faster than the other three algorithms, and TKU-CE+ was the fastest, as shown in Figs. 1(d), (e), and (f), respectively. Both TKU-CE and TKU-CE+ were one order of magnitude faster than TKO on T35I100D7k. On T50I150D10k, TKU-CE was one order of magnitude faster than TKU, TKO, and kHMC; TKU-CE+ was one order of magnitude faster than TKO and kHMC, and two orders of magnitude faster than TKU. On T40I100D20k, TKU-CE was one order of magnitude faster than TKU and kHMC, and two orders of magnitude faster than TKO; TKU-CE+ was one order of magnitude faster than kHMC, and two orders of magnitude faster than TKU and TKO.

To summarize, the two proposed algorithms were always faster than TKU and TKO, and were faster than kHMC in most cases, particularly on the three synthetic datasets. Because CUV-based pruning, sample refinement, and smoothing mutation were used, TKU-CE+ was faster than TKU-CE.

5.3 Memory consumption

In this subsection, the memory usage of the five algorithms is compared. Memory usage was measured using the Java API. For the reason stated in Section 5.2, not all results were plotted for Chess, Chainstore, and Pumsb.

As shown in Fig. 2, the two proposed heuristic algorithms always consumed less memory than the other three algorithms except on the Chainstore dataset, on which the memory usage of kHMC was 27.57% less than that of TKU-CE+. This is because the four threshold-raising strategies and five pruning properties of kHMC proved effective on this dataset.

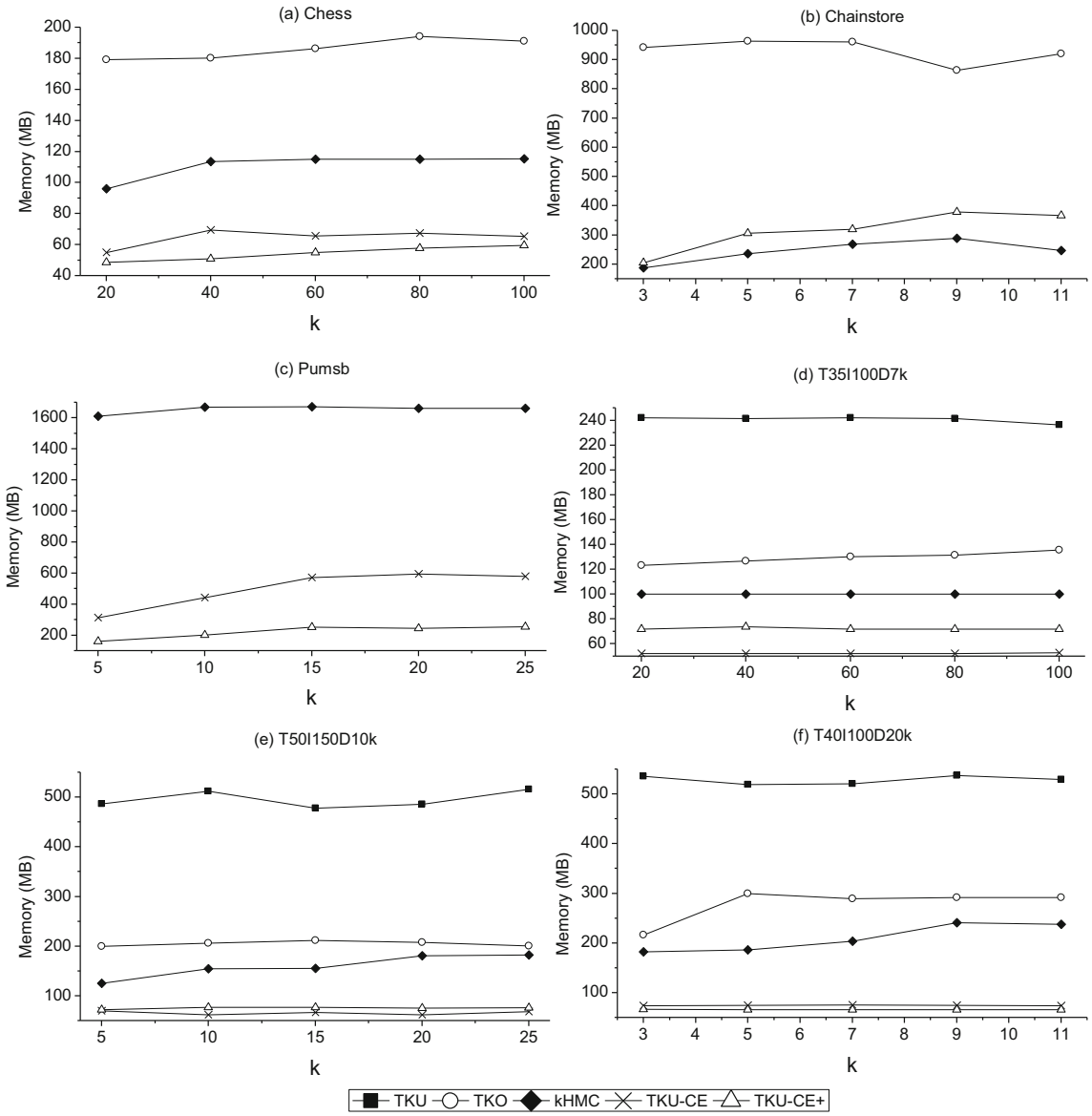


Fig. 2 Memory usage for the six datasets

Of the two heuristic algorithms, TKU-CE+ consumed less memory than TKU-CE on four datasets, but used more memory than TKU-CE on T35I100D7k and T50I150D10k. The reason for this phenomenon is that the pruning strategy based on the CUV had no effect on T35I100D7k and T50I150D10k; this is further verified in Section 5.4. Compared with TKU-CE, TKU-CE+ computed the utility and TWU of each item to perform CUV-based pruning, which led to additional memory usage. This also occurred on the T40I100D20k dataset, shown in Fig. 2(f), where TKU-CE+ consumed less memory than TKU-CE. This is because sample refinement and smoothing mutation were more effective on this synthetic dataset.

Another interesting result is that the amount of memory consumed by both TKU-CE and TKU-CE+ was nearly constant (regardless of the value of k); this demonstrates that the proposed heuristic algorithms have obvious advantages in respect of memory consumption.

To summarize, the two proposed heuristic algorithms consumed less memory than the three exact algorithms, except on Chainstore. Furthermore, the memory usage of TKU-CE and TKU-CE+ was nearly constant. This is because the heuristic algorithms do not use additional tree or list structures for storing and transforming the original information, and do not include any threshold-raising strategies. Thus, compared with the exact algorithms, they are more suitable for the top-k HUIM problem, without the need for a user-specified minimum utility threshold.

Table 6 Lengths of IVs of the two heuristic algorithms

Chess	*k*	20	40	60	80	100
	TKU-CE	75	75	75	75	75
	TKU-CE+	70	72	74	75	75
Chainstore	*k*	3	5	7	9	11
	TKU-CE	46,086	46,086	46,086	46,086	46,086
	TKU-CE+	197	399	463	507	615
Pumsb	*k*	5	10	15	20	25
	TKU-CE	2113	2113	2113	2113	2113
	TKU-CE+	275	291	309	318	350
T35I100D7k	*k*	20	40	60	80	100
	TKU-CE	100	100	100	100	100
	TKU-CE+	100	100	100	100	100
T50I150D10k	*k*	5	10	15	20	25
	TKU-CE	150	150	150	150	150
	TKU-CE+	150	150	150	150	150
T40I100D20k	*k*	3	5	7	9	11
	TKU-CE	100	100	100	100	100
	TKU-CE+	100	100	100	100	100

5.4 Lengths of IVs

As noted in Section 4.4.1, a CUV-based pruning strategy can avoid unpromising itemsets and reduce the length of IVs. To verify the effect of this strategy, we compared the lengths of IVs, and present the results in Table 6.

Table 6 shows that the lengths of IVs of the two algorithms were the same on the three synthetic datasets, and that the IVs of TKU-CE+ were no longer than those of TKU-CE on the three real datasets. The reduction effect (of reducing the lengths of IVs) was most obvious on the two large datasets, Chainstore and Pumsb. Furthermore, the effect was more obvious for smaller values of *k* than for larger values.

5.5 Diversity

To verify the effect of the smoothing mutation proposed in Section 4.4.3, we used the bit edit distance (BED) [29] to evaluate the degree of diversity of the mining results. The edit distance is the minimum number of editing operations—insertion, deletion, or substitution—needed to transform one string to another. $BED(V_i, V_t)$ is defined as

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

where *NBits* is the number of bitwise-complement operations to transform from V_i to V_t.

From the definition of *BED*, the greater the value of $BED(V_i, V_t)$, the greater the diversity between V_i and V_t. For example, transforming V_i = <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, $BED(V_i, V_t) = 3$.

Following [29], two types of BED were used in our experiments. Namely, 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, …, V_N be the IVs in one sample. The Max_BED is defined as

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

As we verified in Section 5.4, the lengths of IVs of the two heuristic algorithms are not always the same. To take account of this, Ave_BED in this set of experiments is defined as

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

Table 7 BED for the Chess dataset

RNI (%)	TKU-CE		TKU-CE+	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
20	0.0560	30	0.0608	32
40	0.0560	22	0.0608	26
60	0.0560	16	0.0608	18
80	0.0439	9	0.0521	11
100	0.0205	6	0.0349	9

Table 8 BED for the Pumsb dataset

RNI (%)	TKU-CE		TKU-CE+	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
20	0.0148	21	0.0144	19
40	0.0105	15	0.0140	13
60	0.0075	11	0.0099	10
80	0.0032	6	0.0086	8
100	0.0019	4	0.0082	7

Table 10 BED for the T50I150D10k dataset

RNI (%)	TKU-CE		TKU-CE+	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
20	0.0280	37	0.0280	39
40	0.0240	8	0.0280	14
60	0.0161	4	0.0176	6
80	0.0124	4	0.0154	4
100	0.0068	2	0.0085	3

where *sizeof(IV)* is the length of IVs. *sizeof(IV)* is the total number of distinct items for TKU-CE, and the actual lengths of IVs for TKU-CE+. This differs from the original definition in [29], which assumes that all vectors have the same length. That is, the definition in Eq. (10) takes account of the diversity in the lengths of the IVs.

Tables 7, 8, 9, 10, and 11 show the comparison results, in terms of BED, on five datasets for different numbers of iterations. We do not report the comparison results on Chainstore because TKU-CE had insufficient memory to run on this dataset. For this set of experiments, we set $k = 10$ for all five datasets. Because TKU-CE and TKU-CE+ did not converge after the same number of iterations, as discussed in Section 5.8, we use the *relative number of iterations* (RNI), which is the percentage of the total number of iterations, for comparison.

Generally, the top-k HUIs discovered by TKU-CE+ had a greater Max_BED and Ave_BED than those discovered by TKU-CE in most cases, for the five datasets. The results demonstrate that the proposed smoothing mutation strategy improved the diversity of the samples. Consequently, execution speed increased and memory usage was reduced.

5.6 Accuracy

A heuristic top-k HUIM algorithm cannot ensure the discovery of all the correct itemsets within a certain number of iterations; that is, some itemsets discovered by TKU-CE and TKU-CE+ may not correspond to the actual top-k HUIs of the entire dataset. In this subsection, we compare the percentage of discovered actual top-k HUIs for different values of k. We used the kHMC algorithm to discover the actual complete list of top-k HUIs from the six datasets. We used the following equation to calculate the accuracy of the top-k HUIs discovered by TKU-CE and TKU-CE+:

$$Acc_k = CE_k/k \times 100\%, \tag{11}$$

where CE_k is the number of actual top-k HUIs discovered by our algorithms.

According to the results in Table 12, disregarding the Chainstore dataset, on which TKU-CE had insufficient memory, the numbers of times that TKU-CE and TKU-CE+ returned 100% of the true top-k HUIs are 17 and 18, respectively. That is, for these five datasets, TKU-CE and TKU-CE+ achieved an accuracy of 100% in 68% and 72% of cases, respectively. In 84% of the cases, TKU-CE achieved an accuracy of no lower than 95%; in 88% of the cases, TKU-CE+ achieved an accuracy of more than 95%. For the Chainstore dataset, TKU-CE+ always returned 100% of the true top-k HUIs. Thus, the results demonstrate that TKU-CE and TKU-CE+ output most of the actual top-k HUIs in a shorter time and using less memory.

Table 9 BED for the T35I100D7k dataset

RNI (%)	TKU-CE		TKU-CE+	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
20	0.0420	31	0.0420	30
40	0.0420	12	0.0308	6
60	0.0241	4	0.0244	6
80	0.0099	3	0.0204	4
100	0.0085	2	0.0184	4

Table 11 BED for the T40I100D20k dataset

RNI (%)	TKU-CE		TKU-CE+	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
20	0.0420	33	0.0420	35
40	0.0410	8	0.0420	11
60	0.0126	4	0.0244	6
80	0.0074	2	0.0231	5
100	0.0067	2	0.0214	4

Table 12 Accuracy for the six datasets

Chess	k		20	40	60	80	100
	TKU-CE	Acc_k (%)	100.00	100.00	100.00	97.50	99.00
	TKU-CE+		100.00	100.00	100.00	98.75	97.00
Chainstore	k		3	5	7	9	11
	TKU-CE	Acc_k (%)	–	–	–	–	–
	TKU-CE+		100	100	100	100	100
Pumsb	k		5	10	15	20	25
	TKU-CE	Acc_k (%)	100	100	100	100	96
	TKU-CE+		100	100	100	100	100
T35I100D7k	k		20	40	60	80	100
	TKU-CE	Acc_k (%)	100.00	95.00	93.33	91.25	88.00
	TKU-CE+		100.00	97.50	93.33	93.75	92.00
T50I150D10k	k		5	10	15	20	25
	TKU-CE	Acc_k (%)	100	100	100	100	100
	TKU-CE+		100	100	100	100	96
T40I100D20k	k		3	5	7	9	11
	TKU-CE	Acc_k (%)	100	100	100	100	90.91
	TKU-CE+		100	100	100	100	100

5.7 Scalability

In the following experiments, we varied the number of items and dataset size to evaluate the scalability of the two proposed algorithms when discovering the top-10 HUIs. The datasets were all generated by the database generator provided in SPMF [9].

We generated a dataset series T30D100k, with the number of items varying from 100 to 500. As shown in Table 13, TKU-CE+ always achieved a higher speed and less memory usage than TKU-CE.

Table 14 shows the scalability of the algorithms when the number of transactions in T20I100 increased from 100,000 to 500,000. Similarly to the results on a varying number of items, TKU-CE+ was always faster than TKU-CE, and consumed less memory, except on the smallest dataset.

From the above discussion, when varying either the number of items or the dataset size, both TKU-CE and TKU-CE+ show approximately linear scalability.

5.8 Convergence

In this subsection, the convergence time is evaluated for all the datasets, to demonstrate the effect of the number of iterations. Because the three exact algorithms TKU, TKO, and kHMC can discover all HUIs, we only plot the convergence performance of the two proposed heuristic algorithms in Fig. 3.

In this set of experiments, as in the others, the performance of TKU-CE on Chainstore was not plotted because of memory overflow. For the other five datasets, we can observe that TKU-CE tends to converge within a smaller number of iterations; although TKU-CE+ requires more iterations to converge, its total convergence time is shorter. The reason for this phenomenon is that TKU-CE+ uses the sample refinement strategy, which often makes the number of IVs in each iteration no greater than the number of IVs in each iteration of TKU-CE. However, because TKU-CE+ takes significantly less time for each iteration than TKU-CE, the total convergence time of TKU-CE+ is shorter.

Table 13 Scalability on T30D100k

	TKU-CE		TKU-CE+	
	Time(Sec)	Memory(MB)	Time(Sec)	Memory(MB)
T30I100D100k	49.96	177.24	29.01	157.22
T30I200D100k	58.09	173.96	30.78	153.00
T30I300D100k	45.29	244.35	38.08	156.28
T30I400D100k	61.92	401.37	35.14	148.70
T30I500D100k	63.57	426.09	32.37	144.19

Table 14 Scalability on T20I100

	TKU-CE		TKU-CE+	
	Time(Sec)	Memory(MB)	Time(Sec)	Memory(MB)
T20I100D100k	51.82	151.87	48.29	188.44
T20I100D200k	174.41	244.89	55.30	239.87
T20I100D300k	123.51	343.68	113.40	283.32
T20I100D400k	154.63	433.86	115.82	414.95
T20I100D500k	160.71	530.61	120.90	487.43

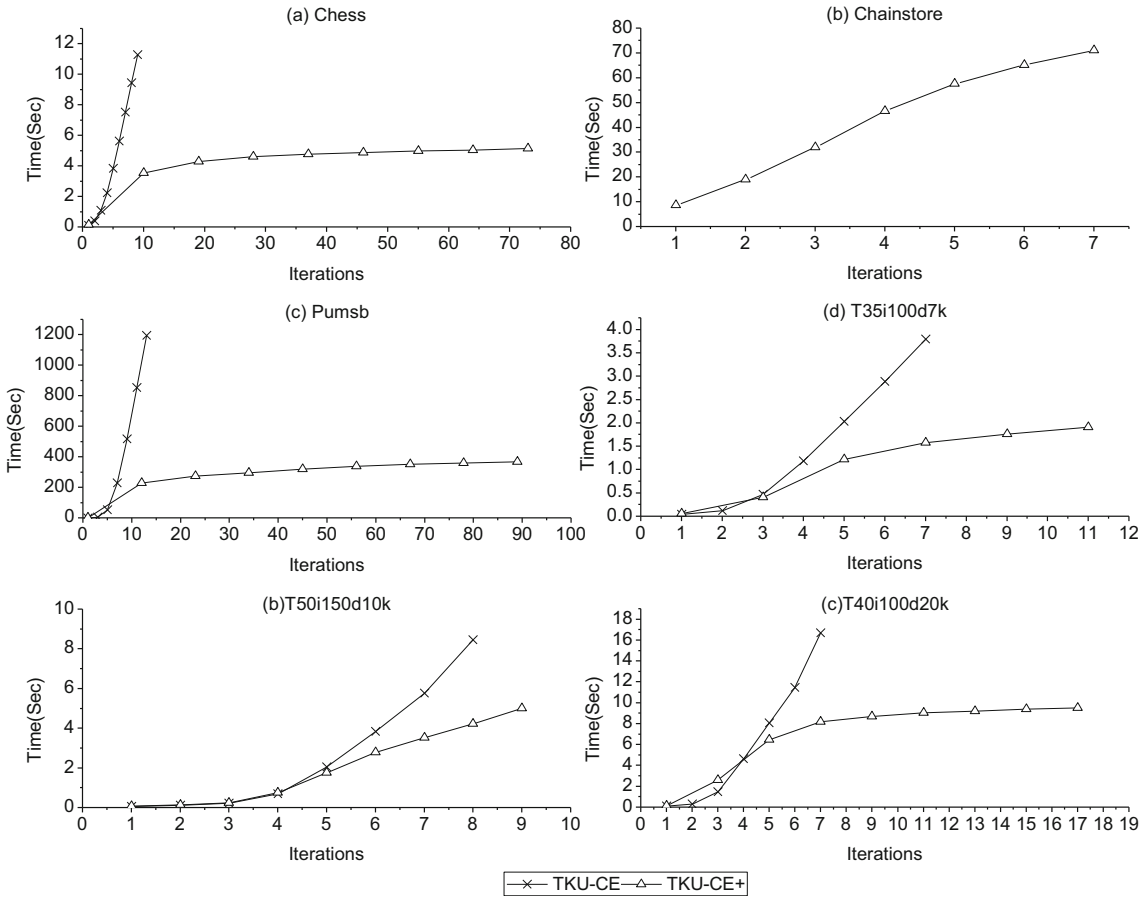


Fig. 3 Convergence performance comparison for the six datasets

6 Conclusions and future work

In this paper, we proposed two CE-based algorithms, called TKU-CE and TKU-CE+, to solve the top-*k* HUIM problem. In contrast to existing exact top-*k* HUIM algorithms, they approach the optimal results using a stochastic method. The two heuristic algorithms do not use additional tree or list structures to represent or transform the original information. Furthermore, the popular threshold-raising strategies are also avoided in the two heuristic algorithms. The experimental results on both synthetic and real datasets demonstrated that TKU-CE and TKU-CE+ are scalable and that they discovered most actual top-*k* HUIs with high speed and low memory usage.

For the two heuristic algorithms, TKU-CE is a baseline algorithm using CE optimization directly, whereas TKU-CE+ improves TKU-CE in three respects. First, TKU-CE+ features a CUV-based pruning strategy to filter unpromising itemsets at first. As a result, the search space of the heuristic algorithm is reduced in the initial stage. Second, TKU-CE+ exploits a sample refinement strategy to reduce the size of samples gradually. Consequently, the search space is reduced in the iterative stage. Furthermore, TKU-CE+ uses a smoothing mutation strategy to improve the diversity of the samples. This helps to solve the problem, common among heuristic algorithms, that they easily fall into locally optimal solutions. With these three improvements, TKU-CE+ outperformed TKU-CE in respect of execution time, memory usage, diversity, accuracy, and scalability.

In contrast to HUIM, top-*k* HUIM does not require the setting of a minimum utility threshold, which is a difficult task for non-expert users. Thus, top-*k* HUIM is promising for a wide range of applications. As we demonstrated in this paper, top-*k* HUIM with heuristic methods can discover most results within an acceptable time. Because of the high computational cost, it will be interesting in future to try to use other heuristic methods for mining top-*k* HUIs. Furthermore, designing more effective pruning strategies for mining top-*k* HUIs heuristically is another topic of our future work.

Acknowledgments This paper is a substantially extended version of our conference paper presented at IEA/AIE 2020. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which helped to improve the quality of this paper. We would also like to thank Dr. Quang-Huy Duong for providing the source code of the kHMC algorithm. This work was supported by the National Natural Science Foundation of China (61977001), Great Wall Scholar Program (CIT&TCD20190305), and Beijing Urban Governance Research Center.

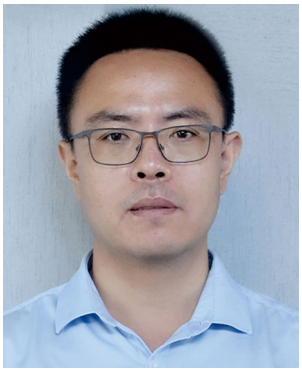
Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

References

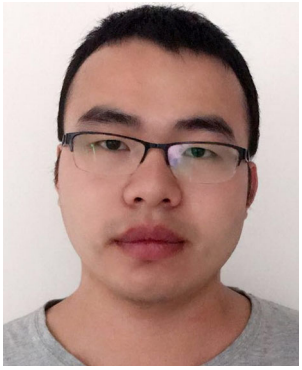
1. Bao R, Yuan X, Chen Z, Ma R (2018) Cross-entropy pruning for compressing convolutional neural networks. Neural Comput 30(11):3128–3149
2. de Boer P-T, Kroese DP, Mannor S, Rubinstein RY (2005) A tutorial on the cross-entropy method. Annals OR 134(1):19–67
3. Dam T-L, Li K, Fournier-Viger P, Duong Q-H (2016) An efficient algorithm for mining top-rank-k frequent patterns. Appl Intell 45(1):96–111
4. Dawar S, Goyal V, Bera D (2017) A hybrid framework for mining high-utility itemsets in a sparse transaction database. Appl Intell 47(3):809–827
5. Deng Z-H (2018) An efficient structure for fast mining high utility itemsets. Appl Intell 48(9):3161–3177
6. Djenouri Y, Comuzzi M (2017) Combining Apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem. Inform Sciences 420:1–15
7. Duong Q-H, Liao B, Fournier-Viger P, Dam T-L (2016) An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. Knowl-Based Syst 104:106–122
8. Fournier-Viger P, Li J, Lin J C-W, Chi T T, Kiran RU (2020) Mining cost-effective patterns in event logs Knowl-Based Syst 191
9. Fournier-Viger P, Lin JCW, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT (2016) The SPMF open-source data mining library version 2. In: proceedings of the 19th European conference on machine learning and knowledge discovery in databases (PKDD'16), pp 36–40
10. Fournier-Viger P, Zhang Y, Lin JC-W, Fujita H, Koh YS (2019) Mining local and peak high utility itemsets. Inform Sciences 481: 344–367
11. Gunawan R, Winarko E, Pulungan R (2020) A BPSO-based method for high-utility itemset mining without minimum utility threshold Knowl-Based Syst:190
12. Joseph AG, Bhatnagar S (2018) An online prediction algorithm for reinforcement learning with linear function approximation using cross entropy method. Mach Learn 107(8–10):1385–1429
13. Kannimuthu S, Premalatha K (2014) Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl Artif Intell 28(4):337–359
14. Kim H, Yun U, Baek Y, Kim J, Vo B, Yoon E, Fujita H (2021) Efficient list based mining of high average utility patterns with maximum average pruning strategies. Inform Sciences 543:85–105
15. Krishna GJ, Ravi V (2020) Mining top high utility association rules using binary differential evolution. Eng Appl Artif Intell 96:103935
16. Krishnamoorthy S (2019) Mining top-k high utility itemsets with effective threshold raising strategies. Expert Syst Appl 117:148–165
17. Lee S, Park J S (2016) Top-k high utility itemset mining based on utility-list structures. In: proceedings of the 2016 international conference on big data and smart computing (BigComp'16), pp 101–108
18. Li X, Yu L, Chang D, Ma Z, Cao J (2019) Dual cross-entropy loss for small-sample fine-grained vehicle classification. IEEE Trans Vehicular Technology 68(5):4204–4212
19. Lin JC-W, Yang L, Fournier-Viger P, Hong T-P, Voznak M (2017) A binary PSO approach to mine high-utility itemsets. Soft Comput 21(17):5103–5121
20. Lin JC-W, Yang L, Fournier-Viger P, Wu JM-T, Hong T-P, Wang S-L L, Zhan J (2016) Mining high-utility itemsets based on particle swarm optimization. Eng Appl Artif Intell 55:320–330
21. Liu Y, Liao W-K, Choudhary A N (2005) A two phase algorithm for fast discovery of high utility of itemsets. In: proceedings of the 9th Pacific-Asia conference on knowledge discovery and data mining (PAKDD'05), pp 689–695
22. Luna JM, Fournier-Viger P, Ventura S (2019) Frequent itemset mining: a 25 years review. Wiley Interdiscip Rev Data Min Knowl Discov 9(6)
23. Nguyen LTT, Vu VV, Lam MTH, Duong TTM, Manh LT, Nguyen TTT, Vo B, Fujita H (2019) An efficient method for mining high utility closed itemsets. Inform Sciences 495:78–99
24. Pazhaniraja N, Sountharrajan S, Kumar BS (2020) High utility itemset mining: a Boolean operators-based modified grey wolf optimization algorithm. Soft Comput 24(21):16691–16704
25. Ryang H, Yun U (2015) Top-k high utility pattern mining with effective threshold raising strategies. Knowl-Based Syst 76:109–126
26. Song W, Huang C (2018) Discovering high utility itemsets based on the artificial bee colony algorithm. In: proceedings of the 22nd Pacific-Asia conference on knowledge discovery and data mining (PAKDD'18), pp 3–14
27. Song W, Huang C (2018) Mining high utility itemsets using bio-inspired algorithms: a diverse optimal value framework. IEEE Access 6:19568–19582
28. Song W, Huang C (2020) Mining high average-utility itemsets based on particle swarm optimization. Data Sci Pattern Recognit 4(2):19–32
29. Song W, Li J (2020) Discovering high utility itemsets using set-based particle swarm optimization. In: proceedings of the 16th international conference on advanced data mining and applications (ADMA'20), pp 38–53
30. Song W, Liu L, Huang C (2020) TKU-CE: cross-entropy method for mining top-k high utility itemsets. In: proceedings of the 33rd international conference on industrial, engineering and other applications of applied intelligent systems (IEA/AIE'20), pp 846–857
31. Song W, Zhang ZH, Li JH (2016) A high utility itemset mining algorithm based on subsume index. Knowl Inf Syst 49(1):315–340
32. Truong T, Duong H, Le B, Fournier-Viger P, Yun U, Fujita H (2021) Efficient algorithms for mining frequent high utility sequences with constraints. Inform Sciences 568:239–264
33. Tseng VS, Wu C-W, Fournier-Viger P, Yu PS (2016) Efficient algorithms for mining top-k high utility itemsets. IEEE Trans Knowl Data Eng 28(1):54–67
34. Vo B, Bui H, Vo T, Le T (2020) Mining top-rank-k frequent weighted itemsets using WN-list structures and an early pruning strategy. Knowl based Syst 201–202
35. Vo B, Nguyen LTT, Nguyen TDD, Fournier-Viger P, Yun U (2020) A multi-core approach to efficiently mining high-utility itemsets in dynamic profit databases. IEEE Access 8:85890–85899

36. Wu C-W, Shie B-E, Tseng V S, Yu P S (2012) Mining top-k high utility itemsets. In: proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'12), pp 78–86
37. Wu JM-T, Srivastava G, Wei M, Yun U, Lin JC-W (2021) Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework. Inform Sciences 553:31–48
38. Wu JM-T, Zhan J, Lin JC-W (2017) An ACO-based approach to mine high-utility itemsets. Knowl-Based Syst 116:102–113
39. Zhang Q, Fang W, Sun J, Wang Q (2019) Improved genetic algorithm for high-utility itemset mining. IEEE Access 7:176799–176813



Wei Song received his Ph.D. degree in Computer Science from University of Science and Technology Beijing, Beijing, China, in 2008. He is a professor in School of Information Science and Technology at North China University of Technology. His research interests are in the areas of data mining and knowledge discovery. He has published more than 40 research papers in refereed journals and international conferences.

Chuanlong Zheng received his B.E. degree in Computer Science and Technology from China University of Mining and Technology, Xuzhou, China, in 2019. Currently, he is pursuing his M.E. degree in School of Information Science and Technology at North China University of Technology, Beijing, China. His research interests include pattern mining, heuristic search and swarm intelligence.



Chaomin Huang received his B.E. degree from Harbin Engineering University, Harbin, China, in 2014, and M.E. degree from North China University of Technology, Beijing, China, in 2019. His research interests are in the areas of data mining and knowledge discovery.

Lu Liu received the B.E. degree in Software Engineering from North University of China, Taiyuan, China, in 2017, and the M.E. degree in Computer Science and Technology from the North China University of Technology, Beijing, China, in 2020. Her research interests include pattern mining and heuristic search.