Discovering High Utility Itemsets Using Set-Based Particle Swarm Optimization

Wei Song(✉) and Junya Li

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 a hot research topic in data mining. Algorithms based on evolutionary computation are attracting increasing attention because they have the advantage of avoiding the combinatorial explosion of the HUI search space. Among evolutionary methods used for mining HUIs, particle swarm optimization (PSO) is the most popular. Existing PSO-based HUI mining (HUIM) algorithms transform positions according to the result of applying the sigmoid function to the velocity. In this paper, we propose an HUIM algorithm based on set-based PSO (S-PSO) called HUIM-SPSO, which mainly considers elements in positions whose velocities are high. We introduce the modeling of HUIM using S-PSO, and explain HUIM-SPSO in detail. To reflect the diversity of the mining results, we propose the measure of the bit edit distance. Extensive experimental results show that the HUIM-SPSO algorithm is efficient and can discover more HUIs with a high degree of diversity.

Keywords: Data mining · High utility itemset · Set-based particle swarm optimization · Bit edit distance

1 Introduction

High utility itemset mining (HUIM) [9, 12] is an extension of frequent itemset mining that is used to discover high-profit itemsets by considering both the quantity and value of a single item. Most existing HUIM algorithms are designed to discover all high utility itemsets (HUIs). Because of the combinatorial explosion incurred by the items in the search space of all HUIs, the performance of exact algorithms tends to quickly degrade with the size of the database and becomes unacceple for large databases. Furthermore, for application fields such as recommender systems, it is not necessary to use all HUIs [14].

Several HUIM approaches have been proposed that use evolutionary algorithms (EAs), such as genetic algorithms (GAs) [3], particle swarm optimization (PSO) [5, 6], and artificial bee colony (ABC) [10]. Although these algorithms can discover reasonable itemsets in an acceptable time, determining how to identify more HUIs within a limited number of iterations is challenging.

PSO is the most widely used EA in previous works [5, 6]; we also consider the HUIM problem from the perspective of PSO. Different from the binary coding scheme PSO [4] used for HUIM, set-based PSO (S-PSO) proposed by Chen et al. [1] is exploited in our algorithm. For S-PSO, the position is updated according to the structure of the cut set, which maintains positions with high speed.

Using S-PSO, we propose an efficient HUIM algorithm called HUIM-SPSO. First, we redefine the S-PSO operations for the HUIM problem. Then, we describe the proposed algorithm in detail. To show the superiority of S-PSO for the HUIM problem, we define the bit edit distance to measure the diversity of the mining results. The experimental results show that HUIM-SPSO is not only efficient but can also discover more HUIs with a high degree of diversity.

2 Preliminaries

2.1 Problem of HUIM

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items. $X \subseteq \boldsymbol{I}$ 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 $TU(T_d) = u(T_d, T_d)$. To mine HUIs, the *minimum utility threshold* δ, which is specified by the user, is defined as a percentage of the total TU values of the database, whereas the *minimum 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 determine all itemsets that have utilities no less than min_util.

The *transaction-weighted utilization* (TWU) of itemset X is the sum of the transaction utilities of all the transactions containing X, which is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$ [9]. X is a high TWU itemset (HTWUI) if $TWU(X) \geq min\_util$. An HTWUI with k items is called a k-HTWUI. It is proved in [9] that HTWUIs satisfy the transaction-weighted downward closure property, and all HUIs are HTWUIs.

Consider the transaction database in Table 1 and the profit table in Table 2. For convenience, we write an itemset $\{C, E\}$ as CE. In the example database, the utility of item E in transaction T_2 is $u(E, T_2) = 3 \times 1 = 3$, the utility of itemset CE in transaction T_2 is $u(CE, T_2) = u(C, T_2) + u(E, T_2) = 6 + 3 = 9$, and the utility of itemset CE in the transaction database is $u(CE) = u(CE, T_2) + u(CE, T_4) = 24$. Given $min\_util = 35$, as $u(CE) < min\_util$, CE is not an HUI. The TU of T_2 is $TU(T_2) = u(ABCDE, T_2) = 18$, and the utilities of other transactions are shown in the third column of Table 1. The TWU of itemset CE is $TWU(CE) = TU(T_2) + TU(T_4) = 48$; thus, CE is an HTWUI.

Table 1. Example database

TID	Transactions	TU
T_1	(B, 1), (C, 3), (D, 5)	35
T_2	(A, 4), (B, 1), (C, 1), (D, 1), (E, 1)	18
T_3	(A, 4), (C, 2), (D, 5)	31
T_4	(C, 2), (D, 5), (E, 1)	30
T_5	(A, 5), (B, 2), (D, 5), (E, 3)	33
T_6	(A, 3), (B, 1), (C, 1), (D, 1)	14
T_7	(D, 1), (E, 1), (F, 2)	8

Table 2. Profit table

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

2.2 Basic Idea of S-PSO

PSO is an EA that imitates the social behavior of bird flocking and fish schooling [4]. In the original PSO algorithm, several particles are initialized at random. Each particle moves toward the optimal value according to the following two equations:

$$Vec_i(t+1) = w \times Vec_i(t) + c_1 \times r_1 \times (PBest_i - Pos_i(t)) + c_2 \times r_2 \times (GBest - Pos_i(t)), \tag{1}$$

$$Pos_i(t+1) = Pos_i(t) + Vec_i(t+1), \tag{2}$$

where $Vec_i(t)$ and $Vec_i(t+1)$ are the velocities of the ith particle at iterations t and $t+1$, respectively; $Pos_i(t)$ and $Pos_i(t+1)$ are the positions of the ith particle at iterations t and $t+1$, respectively; $PBest_i$ is the previous best position of the ith particle; $GBest$ is the current best position of all particles; the three constants w, c_1, and c_2 are weighting coefficients; and r_1 and r_2 are random numbers in the range (0, 1).

All particles update their velocities and positions repeatedly until the best solution is found or the maximum number of iterations is reached.

Velocity Update of S-PSO. In S-PSO, a velocity is a set of probabilities that specify the possibility of each element being used in the position update [1]. Let E be a crisp set of all possible solutions. Vec defined on E is given by

$$Vec = \{p(e)\,|e \in E\}. \tag{3}$$

Using the velocity definition in Eq. 3, the update of the velocity in Eq. 1 is implemented by the following four types of calculation [1].

(1) Coefficient × Velocity: Eq. 1 is composed of three parts. The first part belongs to this type and can be defined as follows:

$$c \times Vec = \{p^*(e)\,|e \in E\}, \tag{4}$$

where

$$p^*(e) = \begin{cases} 1, \text{ if } c \times p(e) > 1 \\ c \times p(e), \text{ otherwise} \end{cases}. \tag{5}$$

(2) Position − Position: In Eq. 1, the second and third parts consist of calculating the differences between the local best position and the current position, and the global best position and the current position. Let Pos_A and Pos_B be two positions. The calculation of the position difference is defined as

$$Pos_A - Pos_B = \{e|e \in Pos_A \wedge e \notin Pos_B\}. \tag{6}$$

(3) Coefficient × (Position − Position): After subtracting the current position from both the local best position and global best position, the calculation of the last two parts is categorized into this type. Similarly, this can also be defined as

$$cE' = \{p'(e)\,|e \in E\}, \tag{7}$$

where

$$p'(e) = \begin{cases} 1, \text{ if } e \in E' \text{ and } c > 1 \\ c, \text{ if } e \in E' \text{ and } 0 \le c \le 1 \\ 0, \text{ if } e \notin E' \end{cases}. \tag{8}$$

(4) Sum of the set of probabilities: The final calculation is the sum of three velocities. Let $Vec_1 = \{p_1(e) \mid e \in E\}$, $Vec_2 = \{p_2(e) \mid e \in E\}$, and $Vec_3 = \{p_3(e) \mid e \in E\}$ be three sets of probabilities defined on E, where the sum of Vec_1, Vec_2, and Vec_3 is defined as

$$Vec_1 + Vec_2 + Vec_3 = \{max(p_1(e), p_2(e), p_3(e))\,|e \in E\}. \tag{9}$$

Position Update of S-PSO. After the velocity update, a random number $\alpha \in (0, 1)$ is generated for each particle. For the jth element of Vec_i, if its corresponding probability $p(e)$ is not smaller than α, then $p(e)$ is retained, that is

$$cut_\alpha(Vec_i^j) = \begin{cases} p(e), \text{ if } p(e) = Vec_i^j \wedge p(e) \ge \alpha \\ 0, \qquad \text{otherwise} \end{cases}. \tag{10}$$

The set of velocity Vec_i, with each element computed by Eq. 10, is called its *cut set*. With the cut set and previous position, the position update in S-PSO is also calculated using Eq. 2.

3 Existing Algorithms

Early HUIM algorithms follow a two-phase routine. The first phase identifies candidate HUIs using the TWU model. Then, the second phase filters the actual HUIs by scanning the original database [9, 12]. Two-phase algorithms typically generate many candidates, which leads to a huge search space and high computational cost. Therefore, one-phase algorithms without candidates were introduced. For one-phase algorithms, data structures, such as a utility list [8] and chain of accurate utility list [7], are used for the efficient discovery of HUIs.

Recently, EAs have been used to traverse immense candidate itemset spaces within an acceptable time to mine HUIs. Two HUIM algorithms, HUPE$_{\text{UMU}}$-GARM and HUPE$_{\text{WUMU}}$-GARM, based on the GA, were proposed in [3]. The difference between them is that the second algorithm does not require a minimum utility threshold. The main problem of these two algorithms is that they tend to fall into local optima easily.

Using ant colony optimization, Wu et al. proposed an HUIM algorithm that generates a routing graph before all the ants start their tours [13]. 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 if it is an HUI.

Song and Huang studied the HUIM problem from the perspective of the ABC algorithm [10]. The proposed HUIM-ABC discovers HUIs by modeling the itemsets as nectar sources. For each nectar source, three types of bees are used for sequential optimization iteratively.

Among various EAs, PSO is the most widely used algorithm in HUIM. HUIM-BPSO$_{\text{sig}}$ [5] and HUIM-BPSO [6] are two PSO-based algorithms for mining HUIs. HUIM-BPSO outperforms HUIM-BPSO$_{\text{sig}}$ using an OR/NOR-tree structure.

In addition to the velocity and previous position used by the binary coding scheme PSO in existing algorithms, the cut set is also used in S-PSO to update positions. Thus, elements corresponding to high velocities tend to have more chances to be retained in the next iteration, which may generate results with high diversity. To the best of our knowledge, S-PSO has not been used in HUIM.

4 Modeling HUIM Using S-PSO

In this paper, each particle is represented by two types of vectors that represent the velocity and position.

Definition 1. Let *SN* be the population size, N_c be the number of 1-HTWUIs, and all 1-HTWUIs be sorted in a total order (e.g., lexicographic order) during the entire mining process. A *velocity vector* V_i $(1 \leq i \leq SN)$ is a vector with N_c elements, and each element V_i^j is a probability, corresponding to the velocity of the *j*th 1-HTWUI, to be used in the position update; and a position vector P_i $(1 \leq i \leq SN)$ is a binary vector with N_c elements, and each element P_i^j is either zero or one, which indicates whether the *j*th 1-HTWUI is absent or present in P_i, respectively.

For these two vectors, a velocity vector changes according to the previous positions, and a position vector then changes with respect to the velocity vector and represents a new candidate itemset. Specifically, if the *j*th position of a position vector contains a one, the item in the *j*th position, according to the total order, is present in a potential HUI; otherwise, this item is not included and cannot be in a potential HUI. For a position vector P_i, its *j*th bit is initialized by either zero or one using roulette wheel selection with the probability

$$p(P_i^j) = \frac{TWU(item_j)}{\sum_{k=1}^{N_c} TWU(item_k)}, \tag{11}$$

where N_c is the number of 1-HTWUIs.

Within each iteration of S-PSO, the fitness function is calculated to characterize the optimization problem. Let X be an itemset represented by a position vector P_i. The utility of X is used as the fitness function directly:

$$fitness(P_i) = u(X). \tag{12}$$

We also need to redefine the calculation of (*Position* – *Position*). Let P_a and P_b be two position vectors with N_c elements. We define

$$dP = P_a - P_b = \{dP_i | \, 1 \leq i \leq N_c\}, \tag{13}$$

where

$$dP_i = \begin{cases} 1, & \text{if } P_a^i = 1 \text{ and } P_b^i = 0 \\ 0, & \text{otherwise} \end{cases}. \tag{14}$$

5 HUIM-SPSO Algorithm

5.1 Bitmap Item Information Representation

We use a bitmap, which is an effective representation of item information in HUIM algorithms [11], in HUIM-SPSO. Specifically, itemsets are represented by a *bitmap cover*. In a bitmap cover, there is one bit for each transaction in the database. If item i appears in transaction T_j, then bit j of the bitmap cover for item i is set to one; otherwise, the bit is set to zero. This naturally extends to itemsets. Let X be an itemset. $Bit(X)$ corresponds to the bitmap cover that represents the transaction set for the itemset X. Let X and Y be two itemsets. $Bit(X \cup Y)$ can be computed as $Bit(X) \cap Bit(Y)$, that is, the bitwise-AND of $Bit(X)$ and $Bit(Y)$. Thus, the utility values of the target itemsets can be calculated efficiently using bitwise operations.

5.2 Proposed Algorithm

According to the above discussion, the proposed HUIM algorithm based on S-PSO (HUIM-SPSO) is shown in Algorithm 1.

Algorithm 1 **HUIM-SPSO**
Input Population size *SN*, minimum utility value *min_util*, maximum number of iterations *max_iter*
Output HUIs

```
1  Init( );
2  $iter = 1$;
3  while $iter \leq max\_iter$ do
4    for $i$=1 to $SN$ do
5      Randomly generate $r_1$ and $r_2$;
6      Calculate $V_i$ using Eq. 1;
7      Randomly generate $\alpha$;
8      for $j$=1 to $N_c$ do
9        if $V_i^j < \alpha$ then
10         $V_i^j = 0$;
11       end if
12     end for
13     Position_update($P_i$);
14     $X = IS(P_i)$;
15     if $u(X) \geq min\_util$ and $X \notin SHUI$ then
16       $X \rightarrow SHUI$;
17     end if
18     $X_l = IS(Pbest_i)$;
19     if $u(X) > u(X_l)$ then
20       $Pbest_i = P_i$;
21     end if
22   end for
23   Find $GBest$ among $SN$ particles;
24   $iter$ ++;
25 end while
26 Output all HUIs in $SHUI$.
```

In Algorithm 1, the initialization procedure (described in Algorithm 2) is called in Step 1. Then, the number of iterations is set to one (Step 2). The main loop (Steps 3–25) repeats the update of the velocity and position vectors until the maximum number of iterations is reached. The loop from Step 4 to Step 22 processes each particle individually. For each particle P_i, Step 5 generates two random numbers in the range (0, 1). It should be noted that we set values of w, c_1, and c_2 in Eq. 1 to one in our algorithm. The velocity vector is calculated in Step 6. Step 7 generates a random number α to modify the velocity vector. Then, the velocity vector V_i of the enumerating particle is updated in the loop from Steps 8–12. The position is updated by calling the procedure described in Algorithm 3. Step 14 determines the itemset that corresponds to the enumerating particle. The function *IS*() returns itemset *X* by unifying the items in P_i if its value is one. The newly determined itemset is stored in *SHUI* if it is an HUI and has not been discovered before. *SHUI* is the set of discovered HUIs. Steps 18–21 update the local best value of the enumerating particle. *GBest* is updated by the particle corresponding to the discovered HUI with the highest utility value in Step 23. Step 24 increments the number of iterations by one. Finally, Step 26 outputs the discovered HUIs.

```
Algorithm 2   Procedure Init( )
Input         Transaction database D, population size SN, minimum utility value
              min_util
Output        The first population of particles
1             Scan database D once;
2             Delete items that are not 1-HTWUIs;
3             Represent the reorganized database as a bitmap;
4             for $i$=1 to $SN$ do
5               for $j$=1 to $N_c$ do
6                 Initialize $P_i^j$ with 0 or 1 using Eq. 11;
7                 if $P_i^j \neq 0$ then
8                   $v_i^j = rand( )$;
9                 end if
10              end for
11              $PBest_i = P_i$;
12              $X = IS(P_i)$;
13              if $u(X) \geq min\_util$ then
14                $X \rightarrow SHUI$;
15              end if
16            end for
17            Find $GBest$ among $SN$ particles.
```

In Algorithm 2, the transaction database is first scanned once to determine the 1-HTWUIs (Steps 1–2). In Step 3, the bitmap representation of the pruned database is constructed. The main loop (Steps 4–16) generates the initial particles individually. The loop from Step 5 to Step 10 initializes each element of the position and velocity vectors of the enumerating particle. The function *rand*() returns a random number in the range (0, 1). Note that we only initialize the elements whose values are one in the position vector with a random velocity. P_i is also initialized as $PBest_i$ in Step 11. Step 12 determines the itemset that corresponds to the enumerating particle. If the current particle can produce an HUI X (Step 13), Step 14 records this itemset. Finally, *GBest* is initialized in Step 17.

```
Algorithm 3  Position_update(P)
Input        Position vector P
Output       New updated position vector P
1  Initialize new_P with all elements equals to 0;
2  Randomly generate a positive number k no higher than $N_c$;
3  if $|V| \geq k$ then
4      Generate new_P by setting k 1s within the positions whose corresponding velocities are not 0;
5  else
6      Generate new_P by setting |V| 1s in the positions whose corresponding velocities are not 0;
7      Change values of $(k-|V|)$-bits of new_P from 0s to 1s;
8  end if
9  P = new_P;
```

In Algorithm 3, the processed position is initialized as a vector with N_c 0 s in Step 1. Then, the new position is built in a constructive manner. Step 2 determines the number of bits to be set to one. The new position first learns from the elements in the corresponding velocity (Steps 3–4). $|V_i|$ is the number of elements whose values are not zeros in V_i. If the construction of *new_P* is not finished by only considering non-zero elements in its velocity vector, other 1-HTWUIs are used to build a new position vector (Steps 5–8). The updated new position is finally determined in Step 9.

5.3 Illustrative Example

We use the transaction database in Table 1 and profit table in Table 2 for the explanation. After the first database scan, the TWU of each item is shown in Table 3.

Table 3. TWU of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*
TWU	96	100	128	169	89	8

Given $min\_util = 35$, as $TWU(F) < min\_util$, item F is deleted from transactions T_7, and the utility of F is eliminated from the TUs of T_7. The reorganized database is then represented by a bitmap, as shown in Table 4.

Assume the size of population *SN* to be 3. As the number of 1-HTWUIs is 5, there are five elements in both the velocity vector and position vector. According to Eq. 11, three position vectors are generated randomly: P_1 = <10111>, P_2 = <11001>, and P_3 = <11010>. Then, three velocities are also generated randomly: $V_1 = \{0.52, 0, 0.87, 0.01, 0.15\}$, $V_2 = \{0.15, 0.58, 0, 0, 0.62\}$, and $V_3 = \{0.76, 0.03, 0, 0.28, 0\}$. For the first population, $PBest_i$ is the same as P_i. Thus, $PBest_1$ = <10111>, $PBest_2$ = <11001>, and $PBest_3$ = <11010>. For the example database, we can see that the three particles represent *ACDE*, *ABE*, and *ABD*, respectively. We have $u(ACDE) = 16$, $u(ABE) = 27$,

Table 4. Bitmap representation of the reorganized database

	A	B	C	D	E
T'_1	0	1	1	1	0
T'_2	1	1	1	1	1
T'_3	1	0	1	1	0
T'_4	0	0	1	1	1
T'_5	1	1	0	1	1
T'_6	1	1	1	1	0
T'_7	0	0	0	1	1

and $u(ABD) = 41$. Among these three itemsets, only *ABD* is an *HUI*, so $SHUI = \{ABD$: $41\}$, where the number after the colon denotes the utility. According to the utility value, *GBest* is $<11010>$.

We then take particle P_1 as an example. According to Eq. 13, $PBest_1 - P_1 = <00000>$ and $GBest - P_1 = <01000>$. Suppose $r_1 = 0.15$ and $r_2 = 0.66$, according to Eq. 7, $r_1(PBest_1 - P_1) = \{0, 0, 0, 0, 0\}$ and $r_2(GBest - P_1) = \{0, 0.66, 0, 0, 0\}$. Then, using Eq. 9, we have the new velocity: $V_1 = \{0.52, 0, 0.87, 0.01, 0.15\} + \{0, 0, 0, 0, 0\} + \{0, 0.66, 0, 0, 0\} = \{0.52, 0.66, 0.87, 0.01, 0.15\}$. Let the randomly generated α be 0.04. For the current V_1, only the fourth element is lower than α, so V_1 is changed to $\{0.52, 0.66, 0.87, 0, 0.15\}$. Then k is randomly generated as one, which indicates that there is only one bit with value one in the new position. Because the first, second, third, and the last elements of V_1 are non-zero, only one of these four bits may be set to one in the new position. Suppose the first bit is selected randomly, the new position vector is $P_1 = <10000>$, which represents itemset *A*. Because $u(A) = 16 < min\_util$, *A* is not an HUI. Furthermore, because $u(A) = u(ACDE)$ and $u(A) < u(ABD)$, neither $PBest_1$ nor *GBest* changes.

In the same iteration, the second and third particles are processed similarly. Then the next iteration starts to process each particle to discover HUIs until the maximal number of iterations is reached.

6 Performance Evaluation

We evaluate the performance of our HUIM-SPSO algorithm and compare it with the HUIM-BPSO$_{\text{sig}}$ [5] and HUIM-BPSO [6] algorithms. We downloaded the source code of the two comparison algorithms from the SPMF data mining library [2].

6.1 Test Environment and Datasets

The experiments were performed on a computer with a 4-core 3.20 GHz CPU and 4 GB memory running 64-bit Microsoft Windows 7. Our programs were written in Java. Four real 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 used for the experimental evaluations

Datasets	Avg. trans. length	No. of items	No. of trans
Chess	37	76	3,196
Mushroom	23	119	8,124
Accidents_10%	34	469	34,018
Connect	43	130	67,557

The four datasets were also downloaded from the SPMF data mining library [2]. The Chess and Connect datasets originate from game steps. The Mushroom dataset contains various species of mushrooms and their characteristics. The Accident dataset is composed of (anonymized) traffic accident data. Similar to the work of Lin et al. [5, 6], only 10% of the total dataset was used for the experiments.

For all experiments, the termination criterion was set to 10,000 iterations and the population size was set to 20.

6.2 Runtime

First, we demonstrate the efficiency performance of these algorithms. When measuring the runtime, we varied the minimum utility threshold for each dataset.

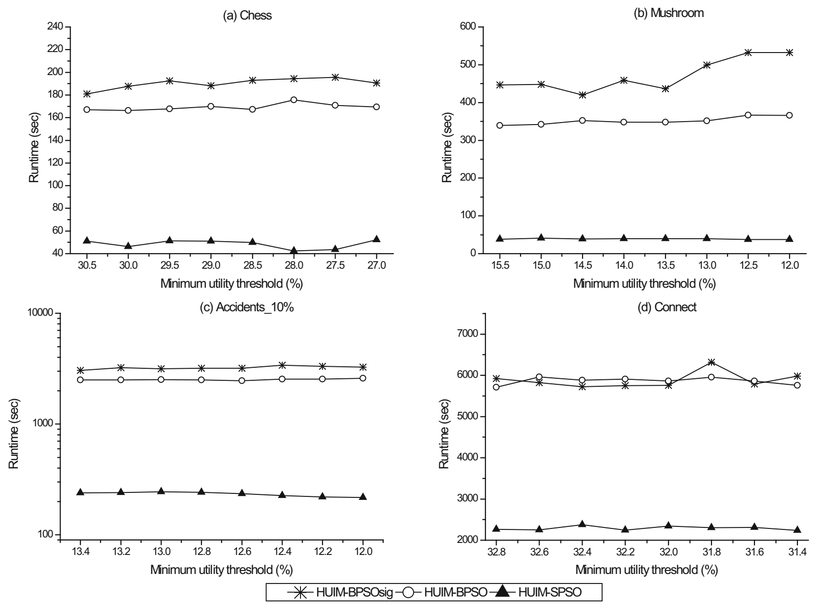

Fig. 1. Execution times for the four datasets

Figure 1(a) compares the execution times for the Chess dataset. We can see that HUIM-SPSO was the most efficient among the three PSO-based HUIM algorithms. On average, HUIM-SPSO was 2.95 times faster than HUIM-BPSO$_{sig}$ and 2.52 times faster than HUIM-BPSO.

For the comparison results on the Mushroom dataset shown in Fig. 1(b), the superiority of the efficiency of the proposed HUIM-SPSO was more obvious. When the minimum threshold changed from 15.5% to 12.0%, HUIM-SPSO always had a steady runtime of approximately 39 s. HUIM-SPSO was one order of magnitude faster than HUIM-BPSO$_{sig}$ and 8.01 times faster than HUIM-BPSO, on average.

From Fig. 1(c), we can see that when the minimum utility threshold changed from 13.4% to 12.0%, HUIM-SPSO was always one order of magnitude faster than HUIM-BPSO$_{sig}$. When the minimum utility threshold was lower than 12.4%, HUIM-SPSO was also one order of magnitude faster than HUIM-BPSO.

When the minimum threshold changed from 32.8% to 31.4% for the Connect dataset, both HUIM-BPSO$_{sig}$ and HUIM-BPSO took more than 5,700 s, and the gap between them was very small, as shown in Fig. 1(d). The proposed HUIM-SPSO algorithm was faster than the above two algorithms, with a runtime constantly close to 2,250 s.

6.3 Number of Discovered HUIs

Because bio-inspired HUIM algorithms cannot ensure the discovery of all itemsets within a certain number of cycles, we compared the number of discovered HUIs among the three PSO-based algorithms.

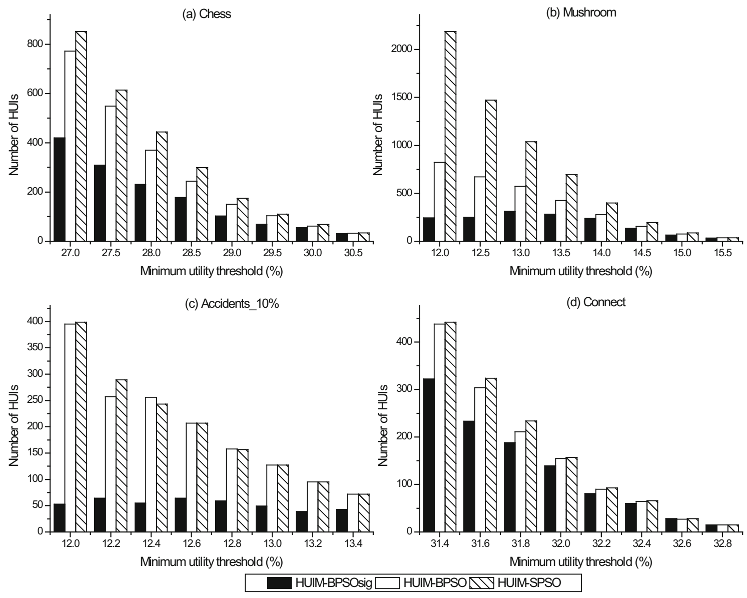

Fig. 2. Number of discovered HUIs for the four datasets

Figure 2 illustrates the number of discovered HUIs for the three algorithms. For all four datasets, the proposed HUIM-SPSO always discovered more HUIs than the other two PSO-based HUIM algorithms, except when the minimum threshold was 12.4% for the Accidents_10% dataset. Generally, the superiority of the number of results of HUIM-SPSO became more obvious as the minimum utility threshold decreased. The maximal gap between HUIM-SPSO and the two counter algorithms appeared for the Mushroom dataset.

6.4 Diversity Comparison

Different from typical optimization problems, such as the traveling salesman problem, which has relatively few best values, all the itemsets with utilities no lower than the minimum threshold are the targets of HUIM. Because the distribution of HUIs is not even, diversity of the population of particles is essential; that is, the higher the diversity within one population, the higher the chance that more results will be discovered within fewer iterations.

To measure the degree of diversity, we propose the *bit edit distance* based on the edit distance used in the field of natural language processing. The edit distance is the minimum number of editing operations – including insertion, deletion, or substitution – needed to transform one string into the other. The bit edit distance (BED) is defined as follows:

$$BED(P, P_t) = NBits, \tag{15}$$

where *NBits* is the number of bitwise-complement operations transformed from position vector P to P_t.

We can see from Eq. 15 that the higher the value of $BED(P, P_t)$, the higher the level of diversity between P and P_t. For example, if we transform $P = <11010>$ to $P_t = <10111>$, three bitwise-complement operations are needed; that is, transform the second bit from 1 to 0, transform the third bit from 0 to 1, and transform the last bit from 0 to 1. Thus, $BED(P, P_t) = 3$.

To measure the diversity of the entire population, we use the pair of position vectors with the highest degree of diversity and average degree of diversity of all pairs of position vectors. Thus, the maximal bit edit distance and average bit edit distance are defined. Let $P_1, P_2, \ldots, P_{SN}$ be position vectors in one population. The maximal bit edit distance (Max_BED) is defined as

$$Max\_BED = max\{BED\left(P_i, P_j\right) | \, 1 \le i \le SN, \ 1 \le j \le SN, i \ne j\}. \tag{16}$$

The average bit edit distance (Ave_BED) is defined as

$$Ave\_BED = \frac{\sum_i \sum_{j \ne i} BED(P_i, P_j)}{SN \times (SN - 1)}. \tag{17}$$

Tables 6, 7, 8 and 9 show the comparison results of bit edit distances for all four datasets for different numbers of iterations. Generally, the HUIs discovered by HUIM-SPSO had a higher Max_BED and Ave_BED than those discovered by HUIM-BPSO for

Table 6. The bit edit distances for the Chess dataset with $\delta = 28.5\%$

Number of iterations	HUIM-BPSO$_{\text{sig}}$		HUIM-BPSO		HUIM-SPSO	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*	*Ave_BED*	*Max_BED*
2000	9.2	18	8.65	14	13.6	31
4000	8.64	14	8.41	15	13.97	32
6000	8.22	13	6.77	11	14.94	36
8000	8.96	16	5.91	12	15.71	36
10000	9.19	16	7.86	15	15.72	36

Table 7. The bit edit distances for the Mushroom dataset with $\delta = 15.5\%$

Number of iterations	HUIM-BPSO$_{\text{sig}}$		HUIM-BPSO		HUIM-SPSO	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*	*Ave_BED*	*Max_BED*
2000	11.83	23	7.05	12	10.52	20
4000	11.21	22	8.37	14	10.44	19
6000	11.30	22	7.81	14	10.46	22
8000	10.06	21	7.58	12	10.96	22
10000	10.14	17	7.62	13	10.18	21

all four datasets. For the other comparison algorithm, HUIM-BPSO$_{\text{sig}}$, the mining results of HUIM-SPSO still had a higher level of diversity on the Chess and Connect datasets, and a comparable level of diversity on the Mushroom and Accidents_10% datasets. For the Mushroom and Accidents_10% datasets, the diversity of the HUIs discovered by HUIM-SPSO was better than that of the HUIs discovered by HUIM-BPSO$_{\text{sig}}$ when the iteration number was high.

Table 8. The bit edit distances for the Accidents_10% dataset with $\delta = 13.0\%$

Number of iterations	HUIM-BPSO$_{\text{sig}}$		HUIM-BPSO		HUIM-SPSO	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*	*Ave_BED*	*Max_BED*
2000	16.95	35	7.09	11	8.6	25
4000	15.98	34	7.80	13	10.19	26
6000	14.44	34	7.33	12	11.13	24
8000	10.50	17	7.29	12	11.13	22
10000	10.83	18	7.62	13	10.84	23

Table 9. The bit edit distances for the Connect dataset with $\delta = 31.6\%$

Number of iterations	HUIM-BPSO$_{sig}$		HUIM-BPSO		HUIM-SPSO	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*	*Ave_BED*	*Max_BED*
2000	7.87	13	6.49	12	17	35
4000	7.85	15	7.82	13	15.53	37
6000	8.39	14	7.33	14	13.07	33
8000	8.31	15	7.55	14	14.64	35
10000	7.44	13	7.74	14	15.19	36

Different from the above two sets of experiments, the diversity measure of HUIM-BPSO$_{sig}$ was better than that of HUIM-BPSO. This is because HUIM-BPSO uses an OR/NOR-tree structure to save valid combinations to discover HUIs. Thus, the prefix structure makes itemsets in the same branch have higher similarity.

7 Conclusions

In this paper, a new HUIM algorithm called HUIM-SPSO was proposed based on S-PSO. In contrast to typical PSO, S-PSO tends to change elements of positions with high velocity rather than simply resorting to sigmoid function transformation. The modeling process of HUIM using S-PSO was described. To measure the diversity of the discovered results, the bit edit distance was proposed. The experimental results demonstrated that the proposed algorithm was efficient and effective.

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. Chen, W.-N., Zhang, J., Chung, H.S.H., Zhong, W.-L., Wu, W.-G., Shi, Y.-H.: A novel set-based particle swarm optimization method for discrete optimization problems. IEEE T. Evolut. Comput. **14**(2), 278–300 (2010)
2. Fournier-Viger, P., et al.: The SPMF open-source data mining library version 2. In: Berendt, B., et al. (eds.) ECML PKDD 2016. LNCS, vol. 9853, pp. 36–40. Springer, Cham (2016)
3. Kannimuthu, S., Premalatha, K.: Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl. Artif. Intell. **28**(4), 337–359 (2014)
4. Kennedy, J., Eberhart, R.: A discrete binary version of particle swarm algorithm. In: Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, pp. 4104–4108 (1997)
5. Lin, J.C.-W., et al.: Mining high-utility itemsets based on particle swarm optimization. Eng. Appl. Artif. Intel. **55**, 320–330 (2016)

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, J., Wang, K., Fung, B.C.M.: Direct discovery of high utility itemsets without candidate generation. In: Proceedings of The 12th IEEE International Conference on Data Mining, pp. 984–989 (2012)
8. Liu, M., Qu, J.-F.: Mining high utility itemsets without candidate generation. In: Proceedings of The 21st ACM International Conference on Information and Knowledge Management, pp. 55–64 (2012)
9. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
10. Song, W., Huang, C.: Discovering high utility itemsets based on the artificial bee colony algorithm. In: Phung, D., Tseng, V., Webb, G.I., Ho, B., Ganji, M., Rashidi, L. (eds.) PAKDD 2018. LNCS (LNAI), vol. 10939, pp. 3–14. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93040-4_1
11. Song, W., Liu, Y., Li, J.: Vertical mining for high utility itemsets. In: Proceedings of the 2012 IEEE International Conference on Granular Computing, pp. 429–434 (2012)
12. Song, W., Zhang, Z., Li, J.: A high utility itemset mining algorithm based on subsume index. Knowl. Inform. Syst. **49**(1), 315–340 (2015). https://doi.org/10.1007/s10115-015-0900-1
13. 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)
14. Yang, R., Xu, M., Jones, P., Samatova, N.: Real time utility-based recommendation for revenue optimization via an adaptive online top-k high utility itemsets mining model. In: Proceedings of The 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, pp. 1859–1866 (2017)