Mining High Average-Utility Itemsets Based on Particle Swarm Optimization

Wei Song Chaomin Huang
College of Computer Science and Technology, North China University of Technology
Beijing 100144
China
songwei@ncut.edu.cn

ABSTRACT

Mining high average-utility itemsets (HAUI) is a promising technique in data mining because it does not favor long itemsets over high utility itemsets (HUIs). Several exact algorithms for mining HAUIs have been proposed. However, as for HUI mining, exact HAUI algorithms tend to require excessive execution times and huge memory space. To overcome this problem, particle swarm optimization (PSO) is applied to the mining of HAUIs. Two algorithms are proposed, one based on the standard PSO algorithm and a second that follows a newly proposed bio-inspired HUI framework. Experimental results show that the former is more efficient, whereas the latter can discover more HAUIs in relatively few iterations.

KEYWORDS

Data mining, High average-utility itemset, Particle swarm optimization, Bit difference set

1 INTRODUCTION

Utility-driven mining (UDM) considers different economic factors during the mining discovery process [6]. As an extension of frequent itemset mining (FIM) [1], high utility itemset mining (HUIM) [12, 15] is an important UDM task that can account for profit and quantity, which are not considered in FIM. The purpose of HUIM is to discover itemsets that will generate high levels of profit. However, longer itemsets will obviously tend to have higher utilities, and so using the same utility threshold will result in itemsets with few items having a low probability of discovery. To normalize the length of high utility itemset (HUI), the problem of high average-utility itemset mining (HAUIM) has been considered [3]. As HAUIM can discover fewer itemsets than HUIM under the same threshold, the problem of HAUIM has received increasing attention.

Hong et al. [3] proposed TPAU, the first algorithm for mining HAUIs. TPAU discovers HAUIs in two phases: the first phase enumerates candidates with an average-utility upper bound (AUUB), and the second phase verifies the actual utilities of candidates to generate the final HAUIs. Similar to the Apriori algorithm for FIM, TPAU follows a level-wise search strategy that usually generates too many candidates and requires multiple database scans.

To reduce the number of candidates and avoid scanning the database multiple times, several methods have been proposed, including the projection-based PAI algorithm [7], tree-structure-based algorithms [11], a method based on the average-utility list [8], and an approach using tighter upper bounds than AUUB [9].

Bio-inspired computation attempts to replicate the way in which biological organisms and sub-organisms operate using abstract computing ideas from living phenomena or biological systems. Several bio-inspired computation methods, such as genetic algorithm [4], particle swarm optimization (PSO) [10], ant colony optimization [16], and artificial bee colony algorithm [14], have been used to explore the huge search space of HUIM. In addition, a bio-inspired computation-based framework (Bio-HUIF) and three HUIM algorithms have recently been proposed [13]. As the main difference between HUI and HAUI is that the latter is not biased toward longer itemsets, the application of bio-inspired algorithms to HAUIM appears promising.

As shown in previous research [10, 13], algorithms based on PSO generally offer excellent performance, and so this study examines the use of PSO to solve the problem of HAUIM. In this paper, two algorithms are proposed. HAUI-PSO is based on the standard PSO approach, while HAUI-PSOD follows the diverse optimal value framework of Bio-HUIF. We describe the two algorithms in detail, and verify their performance experimentally using publicly available datasets. To the best of our knowledge, this is the first attempt to use the PSO algorithm for mining HAUIs.

The remainder of this paper is organized as follows. We describe the problem of HAUIM in Section 2. In Section 3, the application of PSO to HAUIM is discussed in detail. The two PSO-based algorithms are introduced in Sections 4 and 5, respectively. Experimental results are presented and analyzed in Section 6. Finally, we draw our conclusions in Section 7.

2 PROBLEM OF HAUIM

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_M\}$ be a finite set of items. The set $X \subseteq \boldsymbol{I}$ is called an *itemset*; an itemset containing k items is called a k-itemset. Let $\boldsymbol{D} = \{T_1, T_2, \ldots, T_N\}$ be a transaction database. Each transaction $T_i \in \boldsymbol{D}$, with a 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 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 *average-utility* of a k-itemset X in transaction T_d is defined as $au(X, T_d)=u(X, T_d)/k$. The average-utility of X in transaction database $\boldsymbol{D}$ is defined as $au(X) = \sum_{X \subseteq T_d \wedge T_d \in D} au(X, T_d)$.

The *minimum average-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 HAUI if $au(X) \geq min\_util$. Given a transaction database $\boldsymbol{D}$, the task of HAUIM is to determine all itemsets that have average-utilities of no less than min_util.

The *maximal utility* of transaction T_d is defined as $mu(T_d) = max\{u(i_j)|i_j \in T_d\}$. The *average-utility upper bound* (AUUB) of itemset X is defined as $auub(X) = \sum_{X \subseteq T_d \wedge T_d \in D} mu(T_d)$. If $auub(X) \geq min\_util$, X is called a *candidate high average-utility itemset* (CHAUI) [3, 7, 8]. A CHAUI with k items is called a k-CHAUI. The AUUB model is generally used as a downward closure property to improve the mining efficiency of HAUIs.

Table 1: Example Database

Tid	Transaction	*mu*
T_1	(*A*, 2) (*E*, 4) (*F*, 1)	12
T_2	(*A*, 1) (*B*, 1) (*D*, 1) (*E*, 1)	10
T_3	(*A*, 1) (*C*, 1) (*F*, 1)	7
T_4	(*B*, 1) (*D*, 1) (*E*, 2)	10
T_5	(*A*, 2) (*B*, 1) (*C*, 1) (*E*, 5)	15

Table 2: Profit Table

Item	*A*	*B*	*C*	*D*	*E*	*F*
Profit	5	10	7	8	3	1

Consider the transaction database in Table 1 and the profit table in Table 2. For convenience, we denote an itemset $\{A, E\}$ as AE. In the example database, the utility of item E in transaction T_2 is $u(E, T_2) = 1 \times 3 = 3$, the utility of itemset AE in transaction T_2 is $u(AE, T_2) = u(A, T_2) + u(E, T_2) = 5 + 3 = 8$, and the utility of itemset AE in the transaction database is $u(AE) = u(AE, T_1) + u(AE, T_2) + u(AE, T_5) = 22 + 8 + 25 = 55$. The average-utility of AE in the transaction database is $au(AE) = (u(AE, T_1) + u(AE, T_2) + u(AE, T_5))/2 = 27.5$. Given $min\_util = 30$, we have $au(AE) < min\_util$, and therefore AE is not an HAUI. However, when using the same threshold, AE is an HUI because $u(AE) > min\_util$. The maximal utility of T_2 is $mu(T_2) = max\{u(A, T_2), u(B, T_2), u(D, T_2), u(E, T_2)\} = 10$. The maximal utilities of the other transactions are listed in the third column of Table 1. The average-utility upper bound of AE is $auub(AE) = mu(T_1) + mu(T_2) + mu(T_5) = 37$. As $auub(AE) > 30$, AE is a CHAUI.

3 Modeling HAUI Discovery Using PSO

3.1 Basic Idea of PSO

PSO is a bio-inspired algorithm that simulates the foraging behavior of birds or fish [5]. In the PSO algorithm, several particles are initialized at random. Each particle then moves toward the optimal value according to the following two equations:

$$v_i(t+1) = wv_i(t) + c_1 r_1 (pbest_i - ps_i(t)) + c_2 r_2 (gbest - ps_i(t)) \quad (1)$$

$$ps_i(t+1) = ps_i(t) + v_i(t+1) \quad (2)$$

where $v_i(t)$ and $v_i(t+1)$ are the velocities of the ith particle at iterations t and $t+1$, $ps_i(t)$ and $ps_i(t+1)$ are the locations of the ith particle at iterations t and $t+1$, $pbest_i$ is the previous best location of the ith particle, $gbest$ is the current best location of all particles, the three constants w, c_1, c_2 are weighting coefficients, and r_1, 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.

3.2 Particle Encoding

In the proposed algorithm, an *encoding vector* is used to represent the position of each particle. The encoding vector is composed of 0s or 1s corresponding to whether an item is absent or present in a particle. If the corresponding jth position of a particle contains a 1, the item in the jth position is present in a potential HAUI; otherwise, this item is not included and cannot be in a potential HAUI. In the proposed method, the size of the encoding vector representing each particle is equal to the number of 1-CHAUIs in the database, with each bit corresponding to one 1-CHAUI.

To speed up the mining process, the concept of *promising encoding vectors* (*PEVs*) [13] is also used. That is, if no transaction contains the newly generated itemset X according to its encoding vector P_i, X can be ignored and P_i is called an *unpromising encoding vector* (*UPEV*); otherwise, P_i is called a PEV.

3.3 Fitness Evaluation

PSO determines the personal and global best positions, and adjusts the velocity of each particle on each iteration. Within each iteration, a fitness function is calculated to characterizes the optimization problem.

Let X be an itemset represented by an encoding vector P_i. The average-utility of X is used as the fitness function for the HAUIM problem:

$$fitness(P_i) = au(X) \quad (3)$$

3.4 Particle Initialization

Let P_i be the encoding vector of position ps_i of the ith particle and P_i^j be the jth bit of P_i, representing 1-CHAUI $item_j$. P_i^j is initialized to either 0 or 1 using roulette wheel selection with the probability

$$Pr(P_i^j) = \frac{mu(item_j)}{\sum_{k=1}^{N} mu(item_k)} \quad (4)$$

where N is the number of 1-CHAUIs.

3.5 Initialization Algorithm

For both HAUI-PSO and HAUI-PSOD, the initialization phase is the same. This procedure is described in Algorithm 1.

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

In Algorithm 1, the transaction database is scanned once to determine the 1-CHAUIs (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 one by one. The loop from Steps 5–7 initializes each bit of the encoding vector of the enumerating particle. P_i is also initialized as $pbest_i$ in Step 8. Step 9 determines the itemset corresponding to the enumerating particle. Here, the function *IS*( ) returns itemset X by unifying the items in P_i if its value is 1. If the current particle can produce an HAUI X that has not already been discovered (Step 10), Step 11 records this itemset. Here, *SHAUI* is the set of discovered HAUIs. The loop in Steps 13–15 then initializes the velocity of the current particle at random. Here, the function *rand*( ) returns a random number in the range (0, 1). In Step 17, *gbest* is updated by the particle corresponding to the discovered HAUI with the highest average-utility value.

4 HAUI-PSO ALGORITHM

4.1 Particle Update

From the second iteration, the velocity of the particles is updated according to the standard PSO approach in Eq. (1). The encoding vectors of the particle positions are then updated based on a sigmoid function, as used in the BPSO approach [10]:

$$P_i^j(t+1) = \begin{cases} 1 & rand( ) < sig(v_i^j(t+1)) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where $P_i^j(t+1)$ is the jth bit of P_i at iteration (t+1), $v_i^j(t+1)$ is the jth bit of v_i at iteration (t+1), *rand*( ) is a function that generates a random number in the range of (0, 1), and

$$sig(v_i^j(t+1)) = \frac{1}{1+e^{-v_i^j(t+1)}} \quad (6)$$

4.2 Algorithm Description

Algorithm 2 describes the proposed HAUI-PSO algorithm for mining HAUIs.

Algorithm 2	HAUI-PSO
Input	Population size *SN*, minimum average-utility value *min_util*, maximum number of iterations *max_iter*
Output	HAUIs
1	Init( );
2	*iter* = 1;
3	**while** $iter \leq max\_iter$ **do**
4	**for** i=1 to *SN* **do**
5	Update P_i using Eq. (5);
6	Update v_i using Eq. (1);
7	$X = IS(P_i)$;
8	**if** $au(X) \geq min\_util$ **and** $X \notin SHAUI$ **then**
9	$X \rightarrow SHAUI$;
10	**end if**
11	**if** $au(X) > au(pbest_i)$ **then**
12	$pbest_i = P_i$;
13	**end if**
14	**end for**
15	Find *gbest* among *SN* particles;
16	*iter* ++;
17	**end while**
18	Output all HAUIs.

In Algorithm 2, the procedure Init( ) is called in Step 1. Step 2 then initializes the iteration number to 1. The main loop (Steps 3–

17) discovers HAUIs population by population. For each particle, the encoding vector and velocity are updated in Steps 5 and 6, respectively. Step 7 determines the itemset corresponding to the enumerating particle. If the current particle can produce an HAUI X that has not already been discovered (Step 8), Step 9 records this itemset. Steps 11–13 update the *pbest* of the current particle. In Step 15, *gbest* is updated as the particle corresponding to the discovered HAUI with the highest average-utility value. Step 16 updates the iteration number. Finally, all discovered HAUIs are output in Step 18.

5 HAUI-PSOD ALGORITHM

5.1 Bio-HUIF

Bio-HUIF is a newly proposed framework that solves the problem of HUIM using bio-inspired algorithms [13]. The main idea of BIO-HUIF is as follows. Different from standard bio-inspired algorithms, the optimal values of the current population are not definitely retained in the next population—all discovered HUIs are subjected to roulette wheel selection to determine the target of the next population. This improves the average diversity within the population and enhances the efficiency and quality of mining. The effect of Bio-HUIF has been demonstrated using a genetic algorithm, PSO, and the bat algorithm.

5.2 Update of Velocity and Location

Similar to HUIM using Bio-HUIF, we rewrite Eq. (1) as

$$v_i = v_{i1} + v_{i2} + v_{i3} \tag{7}$$

where v_{i1} is always set to 1 and v_{i2}, v_{i3} are calculated as

$$v_{i2} = \lfloor r_1 \mid BitDiff(P_i, pbest_i) \mid \rfloor \tag{8}$$

$$v_{i3} = \lfloor r_2 \mid BitDiff(P_i, gbest) \mid \rfloor \tag{9}$$

where $|BitDiff(P_i, pbest_i)|$ and $|BitDiff(P_i, gbest)|$ are the number of elements in $BitDiff(P_i, pbest_i)$ and $BitDiff(P_i, gbest)$, respectively; $\lfloor r_1 \mid BitDiff(P_i, pbest_i) \mid \rfloor$ and $\lfloor r_2 \mid BitDiff(P_i, gbest) \mid \rfloor$ denote the largest integers that are less than or equal to $r_1|BitDiff(P_i, pbest_i)|$ and $r_2|BitDiff(P_i, gbest)|$, respectively. $BitDiff(P_i, pbest_i)$ and $BitDiff(P_i, gbest)$ are calculated as

$$BitDiff(P_i, pbest_i) = \{k \mid P_i^k \oplus pbest_i^k = 1, 1 \le k \le N\} \tag{10}$$

$$BitDiff(P_i, gbest) = \{k \mid P_i^k \oplus gbest^k = 1, 1 \le k \le N\} \tag{11}$$

where N is the number of 1-CHAUIs, P_i^k is the kth bit of P_i, $pbest_i^k$ is the kth bit of $pbest_i$, $gbest^k$ is the kth bit of *gbest*, and $\oplus$ denotes exclusive disjunction.

The set of *BitDiff* is used to model the difference between two encoding vectors, a concept that was referred to as the *bit difference set* in [13].

For example, let P_1=11010 and P_2=10110 be two encoding vectors. As 11010 ⊕ 10110 = 01100, $BitDiff(C_1, C_2)$ = {2, 3}, that is, P_1 and P_2 differ in their second and third bits.

5.3 Algorithm Description

The proposed HAUI-PSOD algorithm for mining HAUIs is described in Algorithm 3.

Algorithm 3	HAUI-PSOD
Input	Population size *SN*, minimum average-utility value *min_util*, maximum number of iterations *max_iter*
Output	HAUIs
1	Init( );
2	*iter* = 1;
3	**while** *iter* ≤ *max_iter* **do**
4	**for** *i*=1 to *SN* **do**
5	Randomly change one bit of P_i from 0 to 1 or from 1 to 0;
6	Calculate v_{i2} using Eq. (8);
7	Randomly select v_{i2} bits from elements of *BitDiff*, and change v_{i2} bits of P_i from 0 to 1 or from 1 to 0;
8	Calculate v_{i3} of P_i using Eq. (9);
9	Randomly select v_{i3} bits from elements of *BitDiff*, and change v_{i3} bits of P_i from 0 to 1 or from 1 to 0;
10	$X = IS(P_i)$;
11	**if** $au(X) \ge min\_util$ **and** $X \notin SHAUI$ **then**
12	$X \rightarrow SHAUI$;
13	**end if**
14	**if** $au(X) > au(pbest_i)$ **then**
15	$pbest_i = P_i$;
16	**end if**
17	**end for**
18	Determine *gbest* using roulette wheel selection among HAUIs in *SHAUI*;
19	*iter* ++;
20	**end while**
21	Output all HAUIs.

As for HAUI-PSOD, the first two steps of Algorithm 3 call Init( ) and initialize the iteration number to 1. The main loop (Steps 3–20) discovers HAUIs, with the *SN* particles checked one by one in the loop from Steps 4–17. In Steps 5–9, particle P_i randomly changes one bit using the bitwise complement operation, then randomly changes v_{i2} bits from 0 to 1 or from 1 to 0, and finally changes v_{i3} bits from 0 to 1 or from 1 to 0 at random. Steps 10–21 are the same as those in Algorithm 2, except for Step 18. In Step 18, *gbest* of the current population is selected by Eq. (12) using all discovered HAUIs.

$$P_i = \frac{fitness_i}{\sum_{j=1}^{|SHAUI|} fitness_j} \quad (12)$$

That is, particles representing itemsets with higher average-utilities have a higher probability of being selected as the best location in the next population.

6 PERFORMANCE EVALUATION

We now evaluate the performance of our algorithms and compare them with two exact HAUIM algorithms, HAUI-Miner [8] and EHAUPM [9]. The source code for HAUI-Miner and EHAUPM was downloaded from the SPMF data mining library [2].

6.1 Experimental Environment and Datasets

The experiments were performed on a computer with a 4-Core 3.40 GHz CPU and 8 GB memory running 64-bit Microsoft Windows 10. Our programs were written in Java. Four datasets were used to evaluate the performance of the algorithms. The characteristics of the datasets are presented in Table 3.

Table 3: Characteristics of the Datasets

Dataset	Avg.Trans.Len.	#Items	#Trans
Chess	37	76	3,196
Accidents_10%	34	469	34,018
Connect	43	130	67,557
T25N100D50K	25	100	50,000

The first three datasets were downloaded from the SPMF data mining library [2]. The Chess dataset contains game steps from a number of real-life chess matches. The Accident dataset is composed of (anonymized) traffic accident data; similar to the work of Lin et al. [10], only 10% of the total dataset was used in the experiments. The Connect dataset is also derived from game steps. These three datasets already had utility values assigned.

T25N100D50K is a synthetic dataset generated by the IBM data generator [1]. The parameters used in the IBM data generator are T, N, and D, which represent the average item length per transaction, the total number of different items, and the total number of transactions, respectively. T25N100D50K does not provide the utility value or quantity of each item in each transaction. Like typical HUIM algorithms [12, 15], we randomly generated these values for each item in every transaction. The item quantity ranges from 1–5, and the utility follows a lognormal distribution over the range 1.0–10.0.

For all experiments, the termination criterion was set to 1,000 iterations and the initial population size was set to 20.

6.2 Evaluation of the Execution Time

We first evaluate and compare the execution time of the four algorithms under various minimum average-utility thresholds.

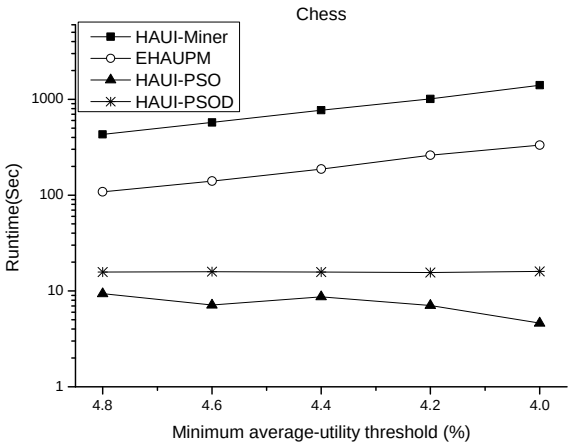

Figure 1: Execution times for the Chess dataset.

Fig. 1 compares the execution times for the Chess dataset. We can see that both HAUI-PSO and HAUI-PSOD are faster than the other two algorithms, with HAUI-PSO being the most efficient. On average, HAUI-PSO is two orders of magnitude faster than HAUI-Miner and one order of magnitude faster than EHAUPM; HAUI-PSOD is one order of magnitude faster than both HAUI-Miner and EHAUPM.

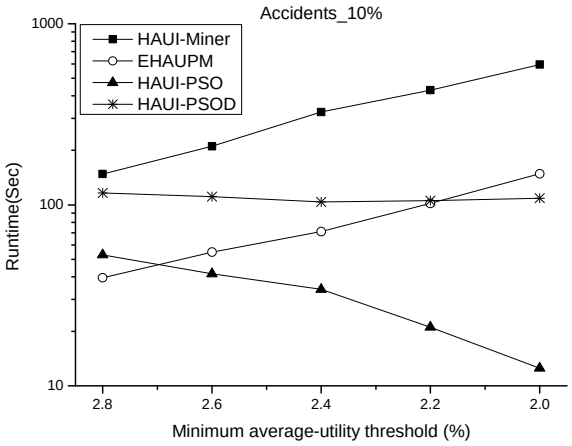

Figure 2: Execution times for the Accidents_10% dataset.

The experimental results using the Accidents_10% dataset are shown in Fig. 2. HAUI-Miner is still the slowest algorithm. EHAUPM shows better performance than with the Chess dataset. In particular, when the minimum average-utility threshold is 2.8%, EHAUPM is faster than the other three algorithms. However, as the minimum average-utility threshold decreases, HAUI-PSO and HAUI-PSOD come to outperform EHAUPM. It is interesting to observe that the execution time of HAUI-PSO does not increase with the decrease of the minimum average-utility threshold. This is because the PSO-based algorithm can not ensure discover all results within certain number of iterations, and the smaller the

minimum average-utility threshold, the fewer number of HAUIs discovered by HAUI-PSO. This can also be verified in Section 6.3.

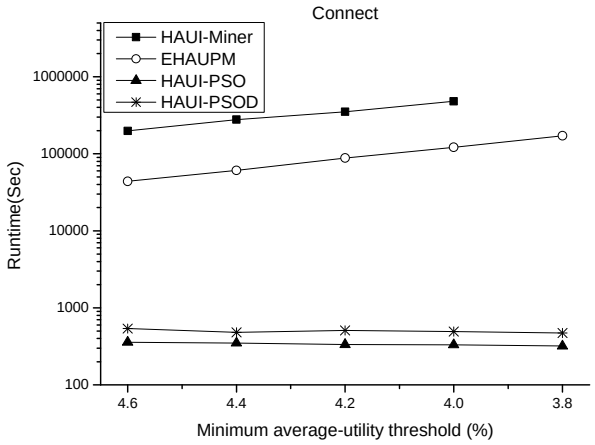

Figure 3: Execution times for the Connect dataset.

The efficiency advantages of the two proposed algorithms are obvious when applied to the Connect dataset (see Fig. 3). Both HAUI-PSO and HAUI-PSOD are two orders of magnitude faster than HAUI-Miner and EHAUPM. When the threshold is 3.8%, HAUI-Miner did not return any results after six days; thus, the result under this threshold is not plotted.

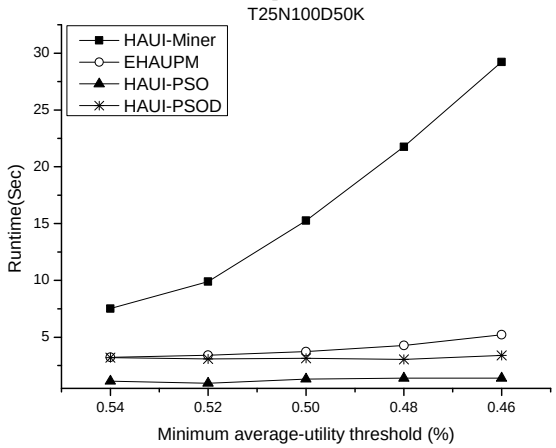

Figure 4: Execution times for the T25N100D50K dataset.

From Fig. 4, we can see that EHAUPM achieves similar efficiency as the two proposed algorithms on the synthetic T25N100D50K dataset. The most efficient algorithm is HAUI-PSO, which is one order of magnitude faster than HAUI-Miner and 2.24 times faster, on average, than EHAUPM.

6.3 Number of Discovered HAUIs

Similar to $HUPE_{UMU}$-GARM [4] and HUIM-BPSO [10], bio-inspired HAUIM algorithms cannot guarantee the discovery of all itemsets within a certain number of cycles. Thus, we compared the percentage of HAUIs discovered by HAUI-PSO and HAUI-PSOD. As HAUI-Miner and EHAUPM are exact algorithms, they would certainly (eventually) find all HAUIs and so we do not present their results. The comparison results are presented in Tables 4–7.

Table 4: Percentage (%) of Discovered HAUIs for the Chess Dataset

Minimum average-utility threshold	HAUI-PSO	HAUI-PSOD
4.0	46.84	86.49
4.2	42.53	97.70
4.4	56.96	98.10
4.6	74.00	100
4.8	75.00	100
Average	59.07	96.46

Table 5: Percentage (%) of Discovered HAUIs for the Accidents_10% Dataset

Minimum average-utility threshold	HAUI-PSO	HAUI-PSOD
2	4.17	64.13
2.2	13.89	84.43
2.4	38.81	95.18
2.6	51.79	98.97
2.8	61.26	100
Average	33.99	88.54

Table 6: Percentage (%) of Discovered HAUIs for the Connect Dataset

Minimum average-utility threshold	HAUI-PSO	HAUI-PSOD
3.8	46.59	96.97
4	57.89	100
4.2	86.21	100
4.4	93.10	100
4.6	100	100
Average	76.76	99.39

Table 7: Percentage (%) of Discovered HAUIs for the T25N100D50K Dataset

Minimum average-utility threshold	HAUI-PSO	HAUI-PSOD
0.46	0	85.63
0.48	0	84.38
0.5	0.13	89.17
0.52	0.16	91.28
0.54	0	91.86
Average	0.06	88.46

The results in these four tables indicate that HAUI-PSOD, although not as efficient as HAUI-PSO, can discover more HAUIs. For example, HAUI-PSOD discovers an average of 99.39% of all the HAUIs in the Connect dataset. For the synthetic dataset T25N100D50K, the gap between the two algorithms is significant, with HAUI-PSOD finding 88.46% of all HAUIs against just 0.06% for HAUI-PSO. This set of experiments shows that Bio-HUIF [13] improves the population diversity of PSO-based HAUIM algorithms. Thus, more results can be discovered compared with using the standard PSO algorithm.

6.4 Convergence

In this subsection, the convergence of our algorithms is evaluated for all the datasets. As the two exact algorithms HAUI-Miner and EHAUPM can discover all HAUIs, we only plot the convergence performance of the two PSO-based HAUIM algorithms. The performance after each iteration is shown in Figs. 5–8.

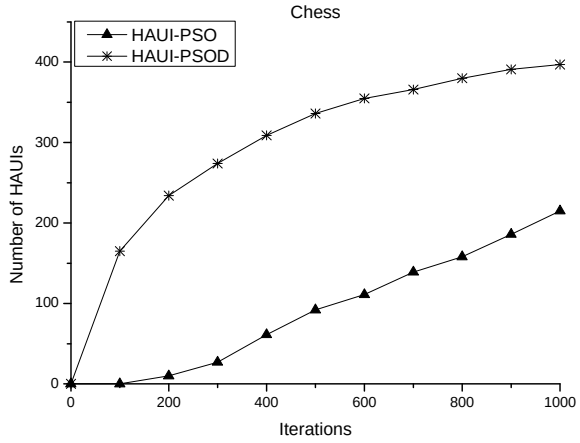

Figure 5: Convergence performance comparison for the Chess dataset.

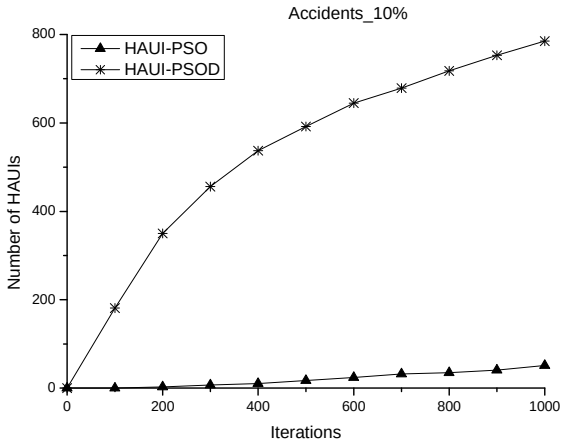

Figure 6: Convergence performance comparison for the Accidents_10% dataset.

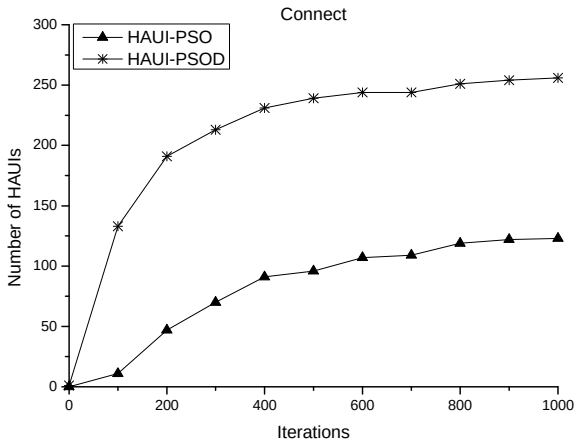

Figure 7: Convergence performance comparison for the Connect dataset.

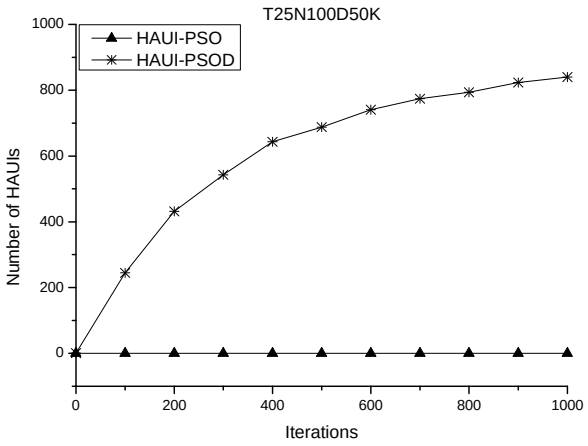

Figure 8: Convergence performance comparison for the T25N100D50K dataset.

These convergence results are consistent with the results for the number of mined HAUIs. That is, HAUI-PSOD always converges faster than HAUI-PSO. The convergence results verify that the search space of HAUIs can be traversed efficiently by randomly changing the optimal values on each iteration using Bio-HUIF. The improved diversity of the population accelerates the convergence speed.

7 CONCLUSION

In this paper, we have examined the problem of HAUIM from the perspective of bio-inspired computation. We proposed two PSO-based algorithms, HAUI-PSO and HAUI-PSOD, for discovering HAUIs. The difference between these two algorithms is that the latter determines the global best value by applying roulette wheel selection to the discovered HAUIs, rather than maintaining the current best value in the next population. We have compared the performance of the proposed algorithms with that of

two exact algorithms, and the experimental results show that the PSO-based approach is a very effective method for HAUIM.

ACKNOWLEDGMENTS

This work was partially supported by Beijing Natural Science Foundation (4162022), and High Innovation Program of Beijing (20150000026833ZK04).

REFERENCES

[1] R. Agrawal, and R. Srikant. Fast algorithms for mining association rules. in *Proceedings of the 20th International Conference on Very Large*, Morgan Kaufmann, 487–499, 1994.

[2] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF open-source data mining library version 2. in *Proceedings of the 19th European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 36-40, 2016.

[3] T.-P. Hong, C.-H. Lee, and S.-L. Wang. Mining high average-utility itemsets. in *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2526-2530, 2009.

[4] S. Kannimuthu, and K. Premalatha. Discovery of high utility itemsets using genetic algorithm with ranked mutation. *Applied Artificial Intelligence*, 28(4): 337-359, 2014.

[5] J. Kennedy, and R. Eberhart. Particle swarm optimization. in *Proceedings of IEEE International Conference on Neural Networks*, IEEE, 1942-1948, 1995.

[6] J. Kleinberg, C. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4): 311–324, 1998.

[7] G. C. Lan, T.-P. Hong, and V. S. Tseng. Efficiently mining high average-utility itemsets with an improved upper-bound strategy. *International Journal of Information Technology & Decision Making*, 11(5): 1009-1030, 2012.

[8] J. C.-W. Lin, T. Li, P. Fournier-Viger, T.-P. Hong, J. Zhan, and M. Voznak. An efficient algorithm to mine high average-utility itemsets. *Advanced Engineering Informatics*, 30: 233-243, 2016.

[9] J. C.-W. Lin, S. Ren, P. Fournier-Viger, and T.-P. Hong. EHAUPM: Efficient high average-utility pattern mining with tighter upper bounds. *IEEE Access*, 5: 12927-12940, 2017.

[10] J. C.-W. Lin, L. Yang, P. Fournier-Viger, T.-P. Hong, and M. Voznak. A binary PSO approach to mine high-utility itemsets. *Soft Computing*, 21(17): 5103–5121, 2017.

[11] T. Lu, B. Vo, H. T. Nguyen, and T.-P. Hong. A new method for mining high average utility itemsets. in *Proceedings of the IFIP International Conference on Computer Information Systems and Industrial Management*, Springer, 33-42, 2014.

[12] W. Song, Z. Zhang, and J. Li. A high utility itemset mining algorithm based on subsume index. *Knowledge and Information Systems*, 49(1): 315-340, 2016.

[13] W. Song, and C. Huang. Mining high utility itemsets using bio-inspired algorithms: a diverse optimal value framework. *IEEE Access*, 6: 19568-19582, 2018.

[14] W. Song, and C. Huang. 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*, Springer, 3-14, 2018.

[15] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(8): 1772-1786, 2013.

[16] J. M.-T. Wu, J. Zhan, J. C.-W. Lin. An ACO-based approach to mine high-utility itemsets. *Knowledge-Based Systems*, 116: 102–113, 2017.