Artificial Fish Swarm Algorithm for Mining High Utility Itemsets

Wei Song(✉), Junya Li, and Chaomin Huang

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

Abstract. The discovery of high utility itemsets (HUIs) is an attractive topic in data mining. Because of its high computational cost, using heuristic methods is a promising approach to rapidly discovering sufficient HUIs. The artificial fish swarm algorithm is a heuristic method with many applications. Except the current position, artificial fish do not record additional previous information, as other related methods do. This is consistent with the HUI mining problem: that the results are not always distributed around a few extreme points. Thus, we study HUI mining from the perspective of the artificial fish swarm algorithm, and propose an HUI mining algorithm called HUIM-AF. We model the HUI mining problem with three behaviors of artificial fish: follow, swarm, and prey. We explain the HUIM-AF algorithm and compare it with two related algorithms on four publicly available datasets. The experimental results show that HUIM-AF can discover more HUIs than the existing algorithms, with comparable efficiency.

Keywords: Data mining · Artificial fish swarm algorithm · High utility itemset · Position vector

1 Introduction

High utility itemsets (HUIs) are extensions of frequent itemsets (FIs) that consider both the unit profit and frequency of occurrence. HUI mining (HUIM) [2] is an active research topic in data mining, and various algorithms [6, 11] for it have been proposed. In contrast to the support measure (which is used in FI mining), utility (used in HUIM) does not satisfy the downward closure property. Hence, the computational cost of HUIM is high. Furthermore, for application fields such as recommender systems, it is not necessary to use all HUIs [13].

To reduce the burden of HUIM, heuristic methods—such as genetic algorithm (GA) [3] and particle swarm optimization (PSO) [5]—have been used for HUIM, to discover acceptable itemsets within a reasonable time. For these algorithms, HUIs are discovered iteratively, and results of one iteration affect the HUIs discovered in the next iteration. Thus, the resulting HUIs tend to be clustered around certain itemsets after many iterations, and the number of results is limited if no new individuals are generated randomly.

Y. Tan and Y. Shi (Eds.): ICSI 2021, LNCS 12690, pp. 407–419, 2021.
https://doi.org/10.1007/978-3-030-78811-7_38

It was also verified in [9] that diversity is of great importance for generating a greater number of HUIs in a smaller number of iterations.

In contrast to GA, PSO, and other heuristic methods that are used for HUIM, the artificial fish swarm algorithm (AFSA) only records the current position, but not other previous information, of each artificial fish (AF). This approach is essentially consistent with the problem of HUIM with a large number of diverse results. Therefore, in this paper, we use the AFSA to formulate an HUIM algorithm. The experiments show that the proposed algorithm can discover more HUIs than two other related algorithms.

2 Preliminaries

2.1 HUIM Problem

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items; each set $X \subseteq \boldsymbol{I}$ is called an *itemset*. Let $\boldsymbol{D} = \{T_1, T_2, \ldots, T_n\}$ be a transaction database. Each transaction $T_d \in \boldsymbol{D}$, with 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 i_p. The *utility* of i_p in 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 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 X in $\boldsymbol{D}$ is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} u(X, T_d)$. The *transaction utility* (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$. The *minimum utility threshold* δ, specified by the user, is a percentage of the total TU values of the database, and the *minimum utility value min_util* $= \delta \times \sum_{T_d \in \boldsymbol{D}} TU(T_d)$. 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 whose utility is 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 [6], and is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} TU(T_d)$. X is a high transaction-weighted utilization itemset (HTWUI) if $TWU(X) \geq min\_util$. An HTWUI containing k items is called a k-HTWUI.

Table 1. Example database.

TID	Transactions	TU
1	(B, 1), (C, 2), (D, 1), (F, 2)	15
2	(A, 4), (B, 1), (C, 3), (D, 1), (E, 1)	18
3	(A, 4), (C, 2), (D,1)	11
4	(C, 2), (D, 1), (E, 1)	11
5	(A, 5), (B, 2), (D, 1), (E, 2)	22
6	(A, 3), (B, 4), (C, 1), (D, 1)	17
7	(D, 1), (E, 1), (F, 1)	12

Table 2. Profit table.

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

As a running example, consider the transaction database in Table 1 and the profit table in Table 2. For convenience, an itemset {B, E} is denoted by BE. The utility of E in T_2 is $u(\text{E}, T_2) = 4 \times 1 = 4$, the utility of BE in T_2 is $u(\text{BE}, T_2) = 2 + 4 = 6$, and the utility of BE in the example database is $u(\text{BE}) = u(\text{BE}, T_2) + u(\text{BE}, T_5) = 18$. Given $min\_util = 35$, because $u(\text{BE}) < min\_util$, BE is not an HUI. The TU of T_2 is $TU(T_2) = u(\text{ABCDE}, T_2) = 18$; the utility of each transaction is shown in the third column of Table 1. Because BE is contained in transactions T_2 and T_5, $TWU(\text{BE}) = TU(T_2) + TU(T_5) = 40$; therefore, BE is an HTWUI.

2.2 Basic Principle of AFSA

The AFSA [4], is inspired by the collective movement of fish and their typical social behaviors. For the AFSA, A_i represents the ith AF. The position of A_i, denoted by $X_i = < x_{i1}, x_{i2}, \ldots, x_{id} >$, represents a possible solution. In addition, the food concentration at position X is denoted by $Y = f(X)$. Let $d_{i,j} = \|X_i - X_j\|$ be the distance between two positions, X_i and X_j. An AF located at X_i inspects the search space around it, within its visual distance (VD). If there is a new position X_j such that $d_{i,j} \leq VD$ and $f(X_j) > f(X_i)$, the AF moves a step toward X_j. To realize this principle, three behaviors of an AF are used iteratively.

Preying Behavior. Preying is the behavior whereby a fish moves to a location with the highest concentration of food. Letting $X_i(t)$ be the current position of the ith AF (A_i) at time t, the position of A_i randomly selected within the visual distance VD is:

$$X_j = X_i(t) + VD \times rand(). \tag{1}$$

where $rand()$ produces a random number between 0 and 1. If $f(X_j) > f(X_i)$, A_i moves a step toward X_j, as follows:

$$X_i(t+1) = X_i(t) + S \times rand() \times \frac{X_j - X_i(t)}{\left\| X_j - X_i(t) \right\|}. \tag{2}$$

where S is the maximum length of a step that an AF can take at each movement. Otherwise, A_i selects a position X_j randomly again, using Eq. 1, and decides whether it satisfies the forward condition. If it cannot satisfy it after *try_number* times, it moves a step randomly, as follows:

$$X_i(t+1) = X_i(t) + VD \times rand(). \tag{3}$$

Swarming Behavior. In nature, a swarm of fish tends to assemble, so as to be protected from danger while avoiding overcrowded areas. Suppose that there are n_f AFs within distance VD of A_i, if the following conditions are satisfied:

$$\begin{cases} f(X_c) > f(X_i) \\ \frac{n_f}{n} < \delta \end{cases}. \tag{4}$$

where n is the total number of AFs, $\delta \in (0, 1)$ is the *crowding factor*, and $X_c = < x_{c1}, x_{c2}, \ldots, x_{cd} >$ is the central position within distance VD of A_i, whose elements are determined by:

$$x_{ck} = \sum_{m=1}^{n_f} x_{mk} \Big/ n_f. \tag{5}$$

where $1 \leq k \leq d$. This means that there is more food in the center, and the area is not overcrowded. Thus, $X_i(t)$ moves a step toward the companion center using:

$$X_i(t+1) = X_i(t) + S \times rand() \times \frac{X_c - X_i(t)}{\|X_c - X_i(t)\|}. \tag{6}$$

If the conditions in Eq. 4 are not satisfied, the preying behavior is executed instead.

Following Behavior. When a fish finds a location with a higher concentration of food, other fish follow. Let X_b be the position within distance VD of A_i with the highest food concentration. That is, for all X_j such that $d_{i,j} \leq VD$, $F(X_j) \leq f(X_b)$. If the following conditions are satisfied:

$$\begin{cases} f(X_b) > f(X_i) \\ \frac{n_f}{n} < \delta \end{cases}. \tag{7}$$

A_i goes forward a step to X_b using:

$$X_i(t+1) = X_i(t) + S \times rand() \times \frac{X_b - X_i(t)}{\|X_b - X_i(t)\|}. \tag{8}$$

Otherwise, A_i executes a preying behavior.

3 Related Work

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. Because HUIM is a task with high computational cost, heuristic methods are suitable for traversing the very large search spaces of HUIM within an acceptable time.

A GA was the first heuristic method used for HUIM, and two HUIM algorithms, $HUPE_{UMU}$-GARM and $HUPE_{WUMU}$-GARM, were proposed in [3]. The difference between them is that the second algorithm does not require a minimum utility threshold. These two algorithms tend to fall into local optima, leading to low efficiency and fewer mining results. Zhang et al. proposed an HUIM algorithm with four strategies [14]—neighborhood exploration, population diversity improvement, invalid combination avoidance, and HUI loss prevention—to improve the algorithm's performance.

PSO is another heuristic method used for HUIM. Lin et al. proposed an HUIM algorithm based on PSO with a binary coding scheme [5]. Song and Li proposed an HUIM algorithm based on set-based PSO [9]. The main difference is that the concept of cut set is used in the latter algorithm to improve the diversity of the resulting HUIs.

Other heuristic methods have also been used for HUIM, including an artificial bee colony (ABC) algorithm [7] and ant colony optimization (ACO) [12]. Furthermore, heuristic algorithms are also used for mining some other specific HUIs, such as top-k HUIs [10] and high average-utility itemsets [8].

4 Modeling HUIM Using AFSA

We use a *position vector* (PV) to represent the position of an AF. Letting HN be the number of 1-HTWUIs, a PV is represented by an HN-dimensional binary vector, in which each bit corresponds to one 1-HTWUI. Assuming that all 1-HTWUIs are sorted in a total order, if the kth 1-HTWUI appears in a PV, then bit k of the PV is set to 1; otherwise, the bit is set to 0. It is proved in [6] that an item with a TWU value lower than the minimum utility threshold cannot appear in an HUI. Thus, only 1-HTWUIs are considered for representation in a PV.

Letting P be a PV, the jth ($1 \leq j \leq NH$) bit of P is randomly initialized using roulette wheel selection with the probability:

$$Pr(P(j)) = \frac{TWU\left(i_j\right)}{\sum_{k=1}^{NH} TWU\left(i_k\right)}. \tag{9}$$

In the same manner as other related work [5, 7], we use the utility of the itemset directly as the object for optimization. Letting X be the itemset corresponding to P,

$$f(P) = u(X). \tag{10}$$

5 The Proposed HUIM-AF Algorithm

5.1 Algorithm Description

Algorithm 1 describes our HUIM algorithm, HUIM-AF.

Algorithm 1	HUIM-AF
Input	Transaction database ***D***, minimum utility value *min_util*, population size *N*, maximum number of iterations *max_iter*, maximum number of attempts *try_number*
Output	HUIs
1	Initialize *N* PVs using Eq. 9;
2	*SHUI* = ∅;
3	*iter* = 1;
4	**while** *iter* ≤ *max_iter* **do**
5	**for** *i*=1 to *N* **do**
6	*is_follow* = false;
7	*is_swarm* = false;
8	P_i = Follow(P_i);
9	**if**(!*is_follow*) **then**
10	Swarm(P_i);
11	**end if**
12	**if**(!*is_follow* AND !*is_swarm*) **then**
13	Prey(P_i);
14	**end if**
15	**end for**
16	*iter* ++;
17	**end while**
18	Output HUIs.

In Algorithm 1, the initial population is generated in Step 1. *SHUI*, the set of discovered HUIs, is initialized as an empty set in Step 2. Step 3 sets the iteration counter to 1. In the loop in Steps 4–17, each AF performs one of the three behaviors, namely follow, swarm, or prey, and then the iteration counter is incremented by 1. This procedure of AFs (performing one action and incrementing the iteration counter) is repeated until the maximum number of iterations is reached. Here, P_i is the PV of A_i in the current population. Finally, Step 18 outputs all discovered HUIs. The procedures for follow, swarm, and prey are described in Algorithms 2, 3, and 4, respectively.

Algorithm 2	Follow(PV P)
1	$Best\_AF = P_1$;
2	**for** i=1 to N **do**
3	Calculate $dis = BitDiff(P, P_i)$ using Eq. 11;
4	**if**($dis \leq VD$ AND $fitness(P_i) > fitness(Best\_AF)$) **then**
5	$Best\_AF = P_i$;
6	**end if**
7	**end for**
8	$dis' = BitDiff(Best\_AF, P)$;
9	**if** ($dis' > 0$) **then**
10	is_follow = true;
11	Randomly generate a positive integer k no higher than dis';
12	Update P by applying bitwise complement operation on k bits of it;
13	**if** ($f(P) \geq min\_util$ AND $IS(P) \notin SHUI$) **then**
14	$IS(P) \rightarrow SHUI$;
15	**end if**
16	**end if**

In Algorithm 2, describing the follow behavior, Steps 1–7 determine the best PV (that with the highest utility value) within distance VD of the enumerating PV. If the best PV discovered is different from the enumerating PV itself, Step 10 sets the follow variable and Steps 11–12 update the enumerating PV by bitwise complement within the bits that are different from the best PV. The operation in Steps 3 and 8, used for calculating the difference between two PVs, is defined as:

$$BitDiff\,(P_i, P_j) = \left|\left\{n | P_i(n) \oplus P_j(n) = 1\right\}\right|, \tag{11}$$

where $P_i(n)$ is the nth bit of P_i, $\oplus$ is the exclusive disjunction operation, and $|S|$ denotes the number of elements in a set S. According to Eq. 11, the distance between two PVs is represented by the number of their corresponding bits that have different values. If the updated PV has a utility value that is no lower than the minimum threshold, and it is not recorded as an HUI, it is stored in Steps 13–15. The function $IS(P)$ returns itemset X including the items whose corresponding bit in P is one.

Algorithm 3	Swarm(PV P)
1	**for** j=1 to NH **do**
2	count_zero[j] = 0;
3	count_one[j] = 0;
4	**end for**
5	**for** i=1 to N **do**
6	Calculate $dis = BitDiff(P, P_i)$ using Eq. 11;
7	**if**($dis \leq VD$) **then**
8	**for** j=1 to NH **do**
9	count_zero[j]++ if $P_i(j)$ is zero;
10	count_one[j] ++ if $P_i(j)$ is one;
11	**end for**
12	**end if**
13	**end for**
14	**for** j=1 to NH **do**
15	**if**(count_one[j] ≥ count_zero[j])
16	$Cen\_AF[j] = 1$;
17	**Else**
18	$Cen\_AF[j] = 0$;
19	**end if**
20	**end for**
21	**if** ($f(Cen\_AF) \geq min\_util$ AND $IS(Cen\_AF) \notin SHUI$) **then**
22	$IS(Cen\_AF) \rightarrow SHUI$;
23	**end if**
24	**if** ($f(P) < f(Cen\_AF)$) **then**
25	is_swarm = true;
26	$dis' = BitDiff(Cen\_AF, P)$;
27	Randomly generate a positive integer k no higher than dis';
28	Update P by applying bitwise complement operation on k bits of it;
29	**if** ($f(P) \geq min\_util$ AND $IS(P) \notin SHUI$) **then**
30	$IS(P) \rightarrow SHUI$;
31	**end if**
32	**end if**

In Algorithm 3, describing the swarm behavior, two binary arrays for determining the center PV, within distance VD of the enumerating PV, are initialized in Steps 1–4. The same two arrays are then updated in the loop in Steps 5–13. According to these two arrays, the center PV is determined in the loop in Steps 14–20. If the center PV represents an HUI that has not been discovered before, this HUI is recorded in Steps 21–23. If the center PV has a higher utility than the current PV, the swarm variable is set in Step 25. In addition, the current PV is updated according to the center PV in Steps 26–28. If the updated PV represents an HUI that has not been discovered before, this HUI is recorded in Steps 29–31.

Algorithm 4	Prey(PV P)
1	flag = false;
2	*times* = 1;
3	**while** *times* ≤ *try_number* **do**
4	Randomly generate a positive integer k no higher than *VD*;
5	Change k bits of P to generate P';
6	**if** ($f(P') \geq min\_util$ AND $IS(P') \notin SHUI$) **then**
7	$IS(P') \rightarrow SHUI$;
8	**end if**
9	**if** ($f(P') > f(P)$) **then**
10	$P = P'$;
11	flag = true;
12	**break**;
13	**end if**
14	*times* ++;
15	**end while**
16	**if** (! flag) **then**
17	Update P by randomly changing several bits of P;
18	**if** ($f(P) \geq min\_util$ AND $IS(P) \notin SHUI$) **then**
19	$IS(P) \rightarrow SHUI$;
20	**end if**
21	**end if**

In Algorithm 4, describing the prey behavior, a flag parameter, representing whether a better position can be found within *try_number*, is initialized to false in Step 1. The number of tries is then initialized to 1 in Step 2. The loop in Steps 3–15 generates new HUIs and updates the current position by trying at most *try_number* times. If no better positions are found in this loop, the current PV is changed by using a bitwise complement operation randomly on several of its bits in Step 17. If the new PV representing an HUI is not discovered before, it is stored in Step 19.

5.2 An Illustrative Example

We use the transaction database in Table 1 and profit table in Table 2 to explain the algorithm. Given $min\_util = 35$, because $TWU(\text{F}) < min\_util$, item F is deleted from transactions T_1 and T_7. Thus, the PV of an AF is a five-dimensional binary vector, in which the first five bits represent A, B, C, D, and E, respectively.

Assume that the size of the population N is 5 and the visual distance *VD* is 3. According to Eq. 9, five PVs are generated randomly: $P_1 = <10110>$, $P_2 = <11010>$, $P_3 = <10111>$, $P_4 = <11001>$, and $P_5 = <00001>$.

Considering P_1, it first performs the follow behavior, in which it is also initialized as *Best_AF*. The PVs within P_1's visual distance are determined to be P_1, P_2, and P_3. According to Eq. 10, $f(P_1) = 32$, $f(P_2) = 41$, and $f(P_3) = 16$, so $Best\_AF = P_2$. According to Eq. 11, $dis' = BitDiff(Best\_AF, P_1) = 2$, so *is_follow* is set to true. Because the second and third bits of P_1 are different from the corresponding bits of *Best_AF*, one of these two bits (e.g., the third bit) is randomly changed to update the new P_1 to $< 10010 >$,

which represents the itemset AD. Because $u(\text{AD}) > min\_util$, $SHUI = \{\text{AD}\}$. Because the first PV performs the follow behavior, the other two behaviors (swarm and prey) are not performed in this iteration.

In the same iteration, the other four PVs are processed similarly. The next iteration then starts to process each PV to discover HUIs until the maximal number of iterations is reached.

6 Performance Evaluation

In this section, we evaluate the performance of our HUIM-AF algorithm and compare it with the HUPE_{UMU}-GARM [3] and $\text{HUIM-BPSO}_{\text{sig}}$ [5] algorithms. We downloaded the source code of the two comparison algorithms from the SPMF data mining library [1].

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 3.

Table 3. 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 [1]. The Chess and Connect datasets originate from game steps. The Mushroom dataset contains various species of mushrooms, and their characteristics. The Accidents dataset is composed of (anonymized) traffic accident data. Similarly to the work of Lin et al. [5], the dataset used for the Accidents_10% experiments contained only 10% of the total dataset.

For all experiments, the population size was set to 20, *try_number* was set to 3, and the termination criterion was set to 10,000 iterations. Furthermore, *VD* was set to a value specific to each dataset, by:

$$VD = \lfloor 0.1 \times NH \rfloor + 1, \tag{12}$$

where $\lfloor 0.1 \times NH \rfloor$ denotes the largest integer that is less than or equal to $(0.1 \times NH)$.

6.2 Execution Time

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

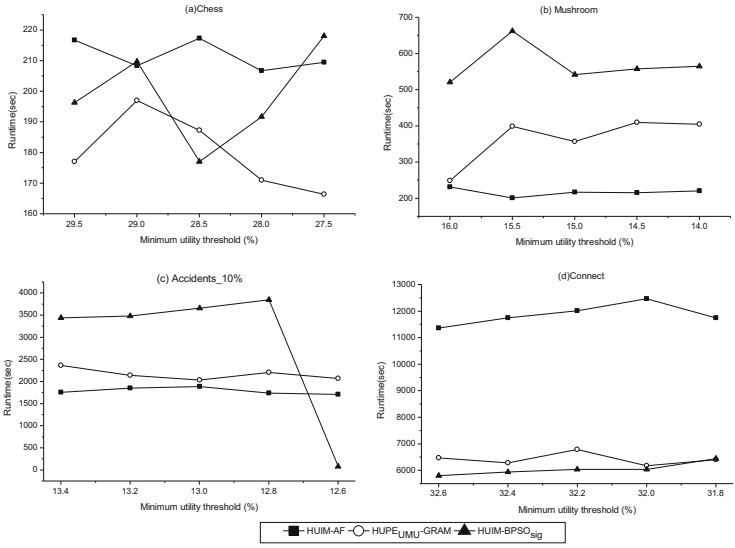

Fig. 1. Execution times for the four datasets.

We can observe from Fig. 1 that the execution time of the proposed HUIM-AF was comparable to that of the other two algorithms. Specifically, it was always faster than the other two algorithms on the Mushroom dataset, whereas its speed was lower than that of the other two algorithms on the Connect dataset. On the Accidents_10% dataset, HUIM-AF was faster than the other two algorithms except when the minimum utility threshold was 12.6%. On the Chess dataset, HUIM-AF was slower than HUPE$_{UMU}$-GARM and comparable to HUIM-BPSO$_{sig}$. The main reason for the speed of HUIM-AF is that the PVs within distance *VD* of the enumerating PV always need to be determined, which incurs a relatively high computational cost.

6.3 Number of Discovered HUIs

Because HUIM algorithms based on heuristic methods cannot ensure the discovery of all itemsets within a certain number of cycles, we compared the number of HUIs discovered by each of the three algorithms. The results are shown in Fig. 2.

In contrast to the results on efficiency, HUIM-AF always discovered more HUIs than the other two algorithms. In particular, there were cases in which both HUPE$_{UMU}$-GARM and HUIM-BPSO$_{sig}$ could not find any results on the Mushroom or Accidents_10% datasets. For example, in the Mushroom dataset, HUPE$_{UMU}$-GARM could not find any results when the minimum thresholds were 15% and 16%, and the same occurred for

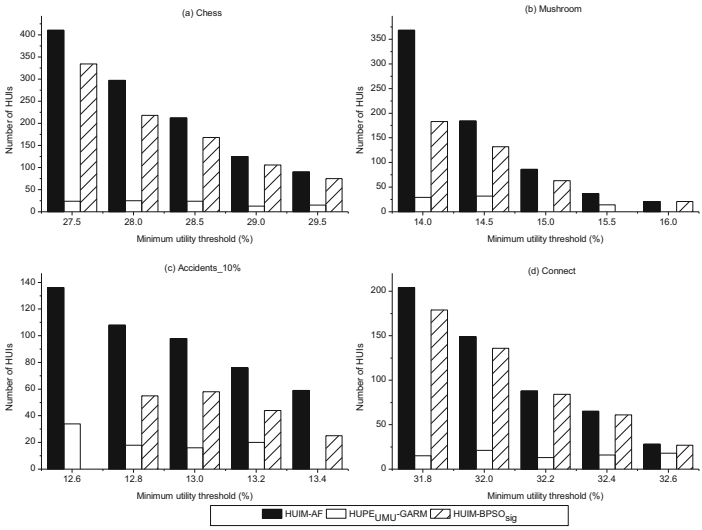

Fig. 2. Number of discovered HUIs for the four datasets.

HUIM-BPSO$_{sig}$ when the minimum threshold was 15.5%. The superiority of HUIM-AF was also demonstrated on Accidents_10% when the threshold was 12.6%. Figure 1 shows that HUIM-BPSO$_{sig}$ was faster than HUIM-AF; however, we can observe from Fig. 2 that HUIM-BPSO$_{sig}$ could not find any results for this threshold.

These experiments show that the AFSA can leap over local optima effectively, so it is more suitable for the HUIM problem because there are multiple targets to optimize.

7 Conclusions

In this paper, we propose an HUIM algorithm following the AFSA paradigm. We formally model the problem in which AFs have three behaviors, and describe the algorithm in detail, with an example. Our experimental results show that the AFSA can discover more HUIs than two other heuristic methods. Our future work includes the design of effective pruning strategies to make this type of algorithm more efficient.

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

References

1. Fournier-Viger, P., et al.: The SPMF open-source data mining library version 2. In: Berendt, B., et al. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9853, pp. 36–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46131-1_8
2. Fournier-Viger, P., Lin, J.C.-W., Nkambou, R., Vo, B., Tseng, V.S. (eds.): High-Utility Pattern Mining. SBD, vol. 51. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-04921-8

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. Li, X., Shao, Z., Qian, J.: An optimizing method based on autonomous animals: fish-swarm algorithm. Syst. Eng. Theor. Pract. **22**(11), 32–38 (2002). (in Chinese)
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. Liu, Y., Liao, W.-K., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
7. Song, W., Huang, C.: Discovering high utility itemsets based on the artificial bee colony algorithm. In: Phung, D., Tseng, V.S., Webb, G.I., Ho, B., Ganji, M., Rashidi, L. (eds.) PAKDD 2018. LNCS (LNAI), vol. 10939, pp. 3–14. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93040-4_1
8. Song, W., Huang, C.: Mining high average-utility itemsets based on particle swarm optimization. Data Sci. Pattern Recogn. **4**(2), 19–32 (2020)
9. Song, W., Li, J.: Discovering high utility itemsets using set-based particle swarm optimization. In: Yang, X., Wang, C.-D., Islam, M.S., Zhang, Z. (eds.) ADMA 2020. LNCS (LNAI), vol. 12447, pp. 38–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65390-3_4
10. Song, W., Liu, L., Huang, C.: TKU-CE: cross-entropy method for mining top-K high utility itemsets. In: Fujita, H., Fournier-Viger, P., Ali, M., Sasaki, J. (eds.) IEA/AIE 2020. LNCS (LNAI), vol. 12144, pp. 846–857. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55789-8_72
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. 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)
13. 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)
14. Zhang, Q., Fang, W., Sun, J., Wang, Q.: Improved genetic algorithm for high-utility itemset mining. IEEE Access **7**, 176799–176813 (2019)