Discovering High Utility Itemsets Based on the Artificial Bee Colony Algorithm

Wei Song(✉) and Chaomin Huang

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

Abstract. Mining high utility itemsets (HUI) is an interesting research problem in data mining. Recently, evolutionary computation has attracted researchers' attention, and based on the genetic algorithm and particle swarm optimization, new algorithms for mining HUIs have been proposed. In this paper, we propose a new algorithm called HUI mining based on the artificial bee colony algorithm (HUIM-ABC). In HUIM-ABC, a bitmap is used to transform the original database that represents a nectar source and three types of bee. In addition to an efficient bitwise operation and direct utility computation, a bitmap can also be used for search space pruning. Furthermore, the size of discovered itemsets is used to generate new nectar sources, which has a higher chance of producing HUIs than generating new nectar sources at random. Extensive tests show that the proposed algorithm outperforms existing state-of-the-art algorithms.

Keywords: Data mining · High utility itemset · Artificial bee colony
Bitmap · Direct nectar source generation

1 Introduction

Unlike frequent itemset [1], high utility itemset (HUI) emphasizes quantity and profit, and has received increasing attention. Although several algorithms [6, 11, 12] have been proposed, enumerating all HUIs exactly cannot avoid the exponential problem of a very large search space when the number of items or size of the database is large.

Evolutionary computation (EC) methods, such as genetic algorithm (GA) [3] and particle swarm optimization (PSO) [9], are applied for mining HUIs recently. Using an EC technique, discovering most HUIs is a promising solution to the problem of the large search space of all HUIs. Two HUI mining algorithms, HUPE_{UMU}-GARM and $\text{HUPE}_{\text{WUMU}}$-GARM, based on the GA are proposed in [8]. The difference between them is that the minimum utility threshold is not required for the second algorithm. Premature convergence is the main problem of these two algorithms; that is, the two algorithms fall easily into local optima. Lin et al. proposed two algorithms, $\text{HUIM-BPSO}_{\text{sig}}$ [5] and HUIM-BPSO [4], for mining HUIs based on PSO. According to [4], HUIM-BPSO outperforms $\text{HUIM-BPSO}_{\text{sig}}$ using an OR/NOR-tree structure.

Unlike the GA and PSO, the artificial bee colony (ABC) algorithm [7] is an algorithm inspired by the foraging behavior of bees. Two distinguishing characteristics of the ABC algorithm are self-organization and decentralized control. To the best of our knowledge, there has not been any work applying ABC algorithm for mining HUIs.

Using the ABC algorithm, we propose a novel HUI mining algorithm called HUI mining based on the ABC algorithm (HUIM-ABC). First, we model the problem of mining HUIs from the perspective of the ABC algorithm. Second, a bitmap is used for both information representation and search space pruning, which can accelerate the HUI discovery process. Third, instead of randomly generating new candidates, the size information of the discovered HUIs is used for producing new nectar sources. Thus, more HUIs can be discovered within fewer iteration cycles. Extensive experimental results show that the HUIM-ABC algorithm outperforms three existing algorithms based on EC in terms of efficiency, number of results, and convergence speed.

2 Preliminaries

2.1 Problem of HUI Mining

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items. Then, 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 $T_i \in \boldsymbol{D}$, 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* (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$.

The *minimum utility threshold* δ, specified by the user, is defined as a percentage of the total TU values of the database, whereas the *minimum 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 HUI mining is to determine all itemsets that have utilities no less than min_util.

The *transaction-weighted utilization* (TWU) of itemset X [6] is the sum of the transaction utilities of all the transactions containing X, which is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$. X is a high transaction-weighted utilization itemset (HTWUI) if $TWU(X) \geq min\_util$. An HTWUI with k items is called a k-HTWUI.

2.2 ABC Algorithm

The main components of the ABC algorithm are nectar sources and artificial bees, and nectar sources (solutions) are refined by the artificial bees iteratively. The value of a nectar source is usually represented by a single number, and there are three types of bee in the ABC algorithm: employed bees, onlooker bees, and scout bees.

Let S_i ($i = 1, 2, \ldots, SN$) be the ith solution with a D-dimensional vector, where SN is the number of nectar sources that are generated randomly initially.

The number of employed bees equals the number of nectar sources. Every employed bee produces a new solution from the old solution using

$$V_{m,j} = S_{m,j} + \varphi\left(S_{m,j} - S_{n,j}\right) \tag{1}$$

where j is a random dimension index in $\{1, 2, \ldots, D\}$, n is a randomly selected nectar source differentiating from m, and φ is a random number in the range $[-1, 1]$. If the fitness value of the newly generated solution is better than that of the old solution, the old solution is forgotten and the new solution is memorized. Otherwise, the old solution is kept.

When all employed bees have finished their searching process, they share the fitness (nectar) information of their solution (nectar sources) with the onlookers. The number of onlooker bees is also *SN*. Each onlooker bee selects one of the memorized nectar sources depending on the fitness value obtained from the employed bees. The probability that a food source will be selected can be obtained from

$$P_i = \frac{fitness_i}{\sum_{j=1}^{SN} fitness_j} \tag{2}$$

where $fitness_i$ is the fitness value of the ith nectar source.

After the nectar source is selected, Eq. 1 is used again by an onlooker bee to generate a new solution. If the fitness value of the new solution is better than that of the old solution, the bee memorizes the new solution and forgets the old solution.

If a nectar source cannot be improved further within a predetermined number of cycles, that nectar source is assumed to be abandoned. Then the nectar source abandoned by the bees is replaced with a new nectar source by one scout bee using

$$S_{m,j} = x_j^{min} + \gamma(x_j^{max} - x_j^{min}) \tag{3}$$

where γ is a random number in [0, 1], and x_j^{min} and x_j^{max} are the lower and upper bounds of dimension j, respectively.

The employed, onlooker, and scout bees' phases repeat until the termination condition is met.

3 Mining HUIs Using the ABC

3.1 Bitmap Item Information Representation

A bitmap is used to transform the original database. The *bitmap* of $\boldsymbol{D}$ is an $n \times m$ Boolean matrix $\boldsymbol{B}(\boldsymbol{D})$ with entries from the set {0, 1}. The entry in $\boldsymbol{B}(\boldsymbol{D})$ that corresponds to transaction T_j $(1 \le j \le n)$ and item i_k $(1 \le k \le m)$ is denoted by (j, k), which is in the jth row and kth column of $\boldsymbol{B}(\boldsymbol{D})$. The value of (j, k) is defined by

$$B_{j,k} = \begin{cases} 1, & \text{if } i_k \in T_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

that is, entry (j, k) of $\boldsymbol{B}(\boldsymbol{D})$ is one if and only if item i_k is included in transaction T_j; otherwise, it is set to zero.

In $\boldsymbol{B}(\boldsymbol{D})$, the *bitmap cover* of item i_k, denoted by $Bit(i_k)$, is the kth column vector. This naturally extends to itemsets: the bitmap cover of itemset X is defined as $Bit(X)$ = bitwise-AND$_{i \in X}(Bit(i))$.

In addition to transforming the original database and representing itemset information, a bitmap can also be used for pruning in the HUIM-ABC algorithm.

Definition 1. Let V be a bit vector that corresponds to a nectar source, employed bee, onlooker bee, or scout bee; and X the itemset that V represents. If $Bit(X)$ is only composed of zeros, V is called an *unpromising bit vector* (UPBV); otherwise, V is called a *promising bit vector* (PBV).

Thus, if a newly generated bit vector is an UPBV, the fitness value computation can be avoided. This technique is called the *PBV check* (PBVC) pruning strategy in the HUIM-ABC algorithm.

3.2 Modeling HUI Discovery Using the ABC

After transforming the database into a bitmap, it is natural to encode each nectar source in a bit vector. The length of this vector is equal to the number of 1-HTWUIs. Based on the bit vector, a new nectar source generation in the HUIM-ABC algorithm can be achieved by only changing the value of 1 bit in the old nectar source, either from zero to one, or from one to zero.

To discover HUIs from the transaction database, the utility of the itemset is used as the fitness function directly:

$$fitness(S_i) = u(X) \tag{5}$$

where X is the union of items in the nectar source S_i if its value is set to one. Thus, X is an HUI if $fitness(S_i) \geq min\_util$.

3.3 Direct Nectar Source Generation for Scout Bees

For the standard ABC algorithm, the scout bees search for a new nectar source randomly. Because the search space of HUIs is huge, when using the simple random search strategy, the number of discovered HUIs could be limited within a certain number of cycles, and the computational cost is high. Thus, the question that arises is: can we generate more promising new nectar sources as early as possible so that more HUIs can be discovered within a certain number of cycles and the computational cost can also be lowered? The answer is "yes" by using the sizes of discovered HUIs.

Park et al. [10] indicated that the processes in the initial iterations of the Apriori algorithm dominate the total execution cost; that is, the candidate sets with smaller sizes (number of items) are crucial to improving the performance of the Apriori algorithm. This inspires us to deduce that the resulting itemsets' sizes are not evenly distributed. Thus, the scout bees can search more frequently in new nectar sources that represent certain sizes that could generate more HUIs. This is called the *direct nectar source generation* (DNSG) strategy in the HUIM-ABC algorithm.

Specifically, we set m buckets, denoted by BK_1, BK_2,…, BK_m, and each bucket stores the number of HUIs whose sizes are within a certain range, denoted by $BK_1.ran$, $BK_2.ran$,…, $BK_m.ran$. For example, the first bucket BK_1 stores the number of HUIs with sizes in $BK_1.ran$ (from 1 to 5), and the second bucket BK_2 stores the number of HUIs with sizes in $BK_2.ran$ (from 6 to 10). Let $BK_1.num$, $BK_2.num$,…., $BK_m.num$ be the number of itemsets in BK_1, BK_2,…, BK_m. Once a new HUI X is generated, and the size of X is in $BK_i.ran$, then $BK_i.num$ is incremented by one. It should be noted that the initial number of $BK_1.num$, $BK_2.num$,…., $BK_m.num$ is set to one because some HUIs with certain sizes may not be discovered during the early cycles. If the initial value is zero, then this type of HUI has no chance of being generated as a new nectar source until the cycle for which the corresponding number is not zero.

For each cycle of the ABC algorithm, after the employed bee phase and onlooker bee phase, the probability of generating new nectar sources whose sizes are in $BK_i.ran$ ($1 \leq i \leq m$) is determined by

$$P^i_{nec} = \frac{BK_i.num}{\sum_{j=1}^{m} BK_j.num} \tag{6}$$

Using the DNSG strategy, the information about the discovered results is used. Thus, the new nectar sources are more promising for HUI discovery than using the simple random approach.

3.4 Algorithm Description

Based on the above discussion, the proposed HUIM-ABC algorithm for mining HUIs is described in Algorithm 1.

Algorithm 1. HUIM-ABC

```
1   Scan database D once. Delete 1-LTWUIs;
2   Represent the reorganized database using a bitmap;
3   Initialization( );
4   gen=1;
5   while gen<=max_cyc do
6       Employed_bees( );
7       Onlooker_bees( );
8       Scout_bees( );
9       gen++;
10  end while
11  Output all discovered HUIs.
```

In Algorithm 1, the transaction database is first scanned once to determine the high TWU single items (Step 1). In Step 2, the bitmap representation of the pruned database is constructed. The procedure Initialization (described in Algorithm 2) is called in Step 3. Then, the number of cycles is set to one (Step 4). The main loop (Steps 5–10) repeats the three phases of the employed bees, onlooker bees, and scout bees until the maximum cycle number is reached. The procedures of these three phases are described in Algorithms 3, 4, and 5, respectively. Finally, Step 11 outputs all discovered HUIs.

```
Algorithm 2. Procedure Initialization( )
1   Initialize $m$ buckets;
2   for each nectar source $S_i$ do
3     $S_i$ is initialized as a bit vector with length |1-HTWUI|;
4     If $S_i$ is an UPBV, generate a new $S_i$ by changing 1 bit until it is a PBV;
5     Get itemset $X$ by unifying of items in $S_i$ if its value is set to 1;
6     while $fitness(S_i) \geq min\_util$ do
7       $X \rightarrow SHUI$;
8       Update $BK_i.num$ $(1 \leq i \leq m)$ using $X$;
9       Initialize a new $S_i$ by changing 1 bit of the original one;
10      If $S_i$ is an UPBV, generate a new $S_i$ by changing 1 bit until it is a PBV;
11      Get itemset $X$ by unifying of items in $S_i$ if its value is set to 1;
12    end while
13    Initialize an employed bee $EB_i$ that corresponds to $S_i$;
14    Initialize an onlooker bee $OB_i$ that corresponds to $S_i$;
15  end for
```

Algorithm 2 first initializes all buckets (Step 1). Then all nectar sources are initialized individually (Steps 2–15). Each source is first represented by a bit vector whose length is equal to the number of 1-HTWUIs (Step 3). Steps 4 performs the PBVC pruning discussed in Sect. 3.1. Step 5 determines the itemset that corresponds to the enumerating nectar source. The loop from Step 6 to Step 12 repeats until a new nectar source with a fitness value lower than the minimum utility value is generated. Step 7 records the newly discovered itemset. Here *SHUI* is the set of discovered HUIs. Step 8 updates the number of one bucket. A new nectar source is generated by randomly changing 1 bit of the original vector (Step 9). Steps 10 also performs PBVC pruning. Step 11 determines the itemset that corresponds to the enumerating nectar source. The employed bees and onlooker bees are then initialized in Steps 13 and 14, respectively.

Algorithm 3 is used by the employed bees to generate the HUIs. The main loop from Step 1 to Step 18 processes all nectar sources individually. In Step 2, *count*, which is a parameter to indicate whether the loop (Steps 7–14) is executed, is set to zero. Then, an employed bee EB_i is mapped to the enumerating nectar source S_i in Step 3. A new employed bee is generated in Step 4. Step 5 performs PBVC pruning. Step 6 determines the itemset X that corresponds to EB_i. The loop from Steps 7–14 repeats until a new employed bee with a fitness value lower than the minimum utility value is generated. Step 8 records the newly discovered HUI. Step 9 updates the number of the bucket whose range covers the size of X. Step 10 generates a new employed bee. Step 11 also performs PBVC pruning. Step 12 determines the itemset X that corresponds to EB_i. Then the parameter *count* is incremented by one in Step 13. If the loop (Steps 7–14) is executed, then S_i is updated by the newest EB_i (Steps 15–17); otherwise, $S_i.trial$, which is a parameter that indicates the number of trials that fail to generate a better nectar source, is incremented by one in Step 17.

Algorithm 3. Procedure Employed_bees( )

```
for $i$=1 to $SN$ do
  count=0;
  Map $EB_i$ to $S_i$;
  Generate a new $EB_i$ by changing 1 bit of the original one;
  If $EB_i$ is an UPBV, generate a new $EB_i$ by changing 1 bit until it is a PBV;
  Get itemset $X$ by unifying of items in $EB_i$ if its value is 1;
  while $fitness(EB_i) \geq min\_util$ do
    $X \rightarrow SHUI$;
    Update $BK_i.num$ ($1 \leq i \leq m$) using $X$;
    Generate a new $EB_i$ by changing 1 bit of the original one;
    If $EB_i$ is an UPBV, generate a new $EB_i$ by changing 1 bit until it is a PBV;
    Get itemset $X$ by unifying of items in $EB_i$ if its value is 1;
    count++;
  end while
  if count>0 then
    Maps $S_i$ to $EB_i$;
  else $S_i$.trial++;
end for
```

Algorithm 4. Procedure Onlooker_bees ( )

```
for $i$=1 to $SN$ do
  count=0;
  Map $OB_i$ to $S_j$ using Eq.2;
  Generate a new $OB_i$ by changing 1 bit of the original source;
  If $OB_i$ is an UPBV, generate a new $OB_i$ by changing 1 bit until it is a PBV;
  Get itemset $X$ by unifying of items in $OB_i$ if its value is 1;
  while $fitness(OB_i) \geq min\_util$ do
    $X \rightarrow SHUI$;
    Update $BK_i.num$ ($1 \leq i \leq m$) using $X$;
    Generate a new $OB_i$ by changing 1 bit of the original one;
    If $OB_i$ is an UPBV, generate a new $OB_i$ by changing 1 bit until it is a PBV;
    Get itemset $X$ by unifying the items in $OB_i$ if its value is 1;
    count++;
  end while
  if count>0 then
    Maps $S_j$ to $OB_i$;
  else $S_j$.trial++;
end for
```

Algorithm 4 is used by the onlooker bees to generate the HUIs of the HUIM-ABC algorithm. It is similar to Algorithm 3, except for Step 3; that is, instead of a one-to-one mapping between employed bees and nectar sources, an onlooker bee maps to a nectar source using the roulette wheel selection mechanism.

Algorithm 5. Procedure Scout_bees( )

```
1  for $i$=1 to $SN$ do
2    if $S_i.trial \geq LT$ then  // $LT$ is the maximum number of trials
3      Initialize a new nectar source $S_i$ using the DNSG strategy;
4      If $S_i$ is an UPBV, generate a new $S_i$ by changing 1 bit until it is a PBV;
5      Get itemset $X$ by unifying of items in $S_i$ if its value is 1;
6      while $fitness(S_i) \geq min\_util$ do
7        $X \rightarrow SHUI$;
8        Update $BK_i.num$ ($1 \leq i \leq m$) using $X$;
9        Generate a new $S_i$ by changing 1 bit of the original one;
10       If $S_i$ is an UPBV, generate a new $S_i$ by changing 1 bit until it is a PBV;
11       Get itemset $X$ by unifying of items in $S_i$ if its value is 1;
12     end while
13   end if
14 end for
```

Algorithm 5 examines the nectar sources individually. If the number of trial times of one nectar source reaches the maximum number of trials (Step 2), then Step 3 uses the DNSG discussed in Sect. 3.3 to generate a new nectar source. Steps 4–12 are similar to the corresponding steps in Algorithms 3 and 4, with the function of recording the discovered HUIs and generating a new promising nectar source.

4 Performance Evaluation

We evaluate the performance of our HUIM-ABC algorithm and compare it with the HUPE$_{\text{UMU}}$-GARM [8], HUIM-BPSO$_{\text{sig}}$ [5], and HUIM-BPSO [4] algorithms.

4.1 Experimental Environment and Datasets

The experiments were performed on a supercomputer with 16-Core 2.00 GHz CPU, 48 GB memory, and running on 64-bit Microsoft Windows 7. Our programs were written in Java. Four datasets, downloaded from the SPMF data mining library [2], were used for evaluation, and their characteristics are presented in Table 1.

Table 1. Characteristics of the datasets

Dataset	Average transaction length	Number of items	Number of transactions
Chess	37	76	3,196
Mushroom	23	119	8,124
Accidents_10%	34	469	34,018
Connect	43	130	67,557

Similar to the work in [4], only 10% of the total Accident dataset was used for our experiment. For all experiments, the termination criterion was set to 2,000 iterations, the number of nectar sources was set to 10, and the number of buckets was 10.

4.2 Running Time

Figure 1 shows the execution time comparisons for the four datasets.

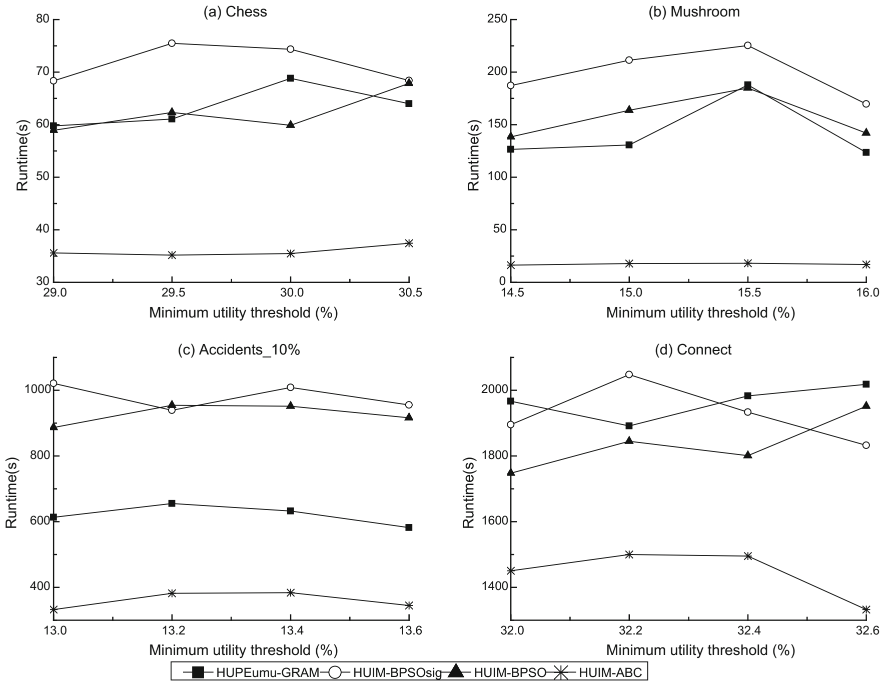

Fig. 1. Execution times for the four datasets

As shown in Fig. 1, the HUIM-ABC algorithm was always faster than the other three algorithms. In particular, the HUIM-ABC algorithm demonstrated relatively steady execution times on the Mushroom dataset, it was 7.19 times faster than HUPE_{UMU}-GARM, 8.06 times faster than HUIM-BPSO, and had an order of magnitude faster than $\text{HUIM-BPSO}_{\text{sig}}$, on average, for this dataset. The reason for the high performance of the HUIM-ABC algorithm can be explained by the use of bitmap representation. In addition to efficient bitwise operations, the PBVC pruning strategy can avoid the unnecessary computation of fitness values as early as possible. Furthermore, the real utility of PBVs can be verified from the recorded transactions rather than by resorting to using the entire database.

4.3 Number of Discovered HUIs

Because the EC-based HUI mining algorithm cannot ensure the discovery of all itemsets within a certain number of cycles, we also compared the number of discovered HUIs. The Two-Phase algorithm [6] was used to discover the actual and complete HUIs from the four datasets. The comparison results are shown in Fig. 2.

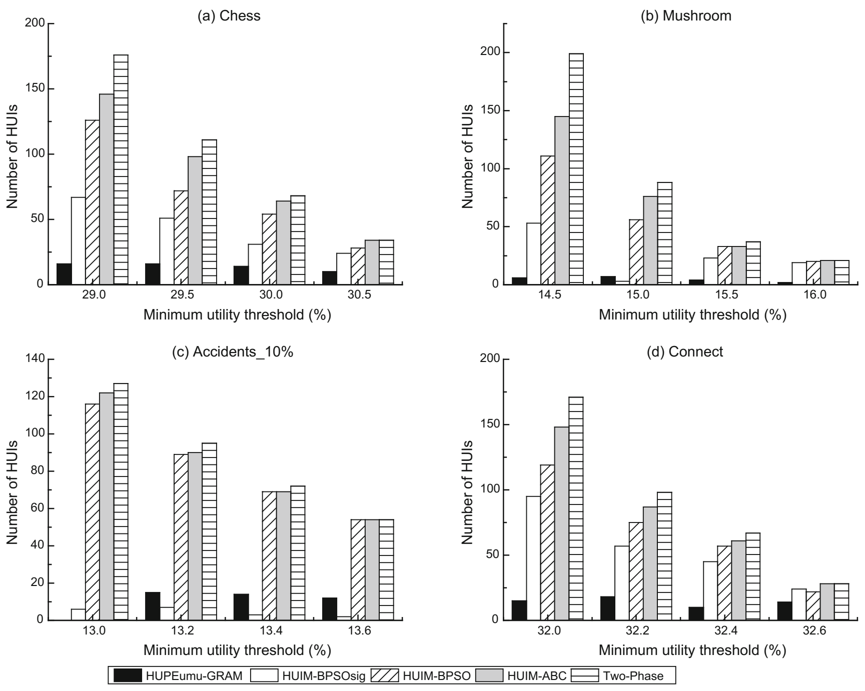

Fig. 2. Number of discovered HUIs

As shown in Fig. 2, the HUIM-ABC algorithm always discovered more HUIs than the other three EC-based algorithms. On average, the HUIM-ABC algorithm discovered 91.34%, 87.10%, 96.65%, and 91.59% of the total number of HUIs on the Chess, Mushroom, Accidents_10%, and Connect datasets, respectively. When the minimum utility threshold was high, for example, 30.5% for Chess, the HUIM-ABC algorithm discovered all the HUIs.

4.4 Convergence

The convergence performance results of the four algorithms are shown in Fig. 3.

For this set of experiments, we can observe that the convergence speed of HUPE_{UMU}-GARM was lower than that of the other three EC-based algorithms. This is because the GA-based algorithm suffered from the combination explosion problem in the evolution process composed of selection, crossover, and mutation. For the two PSO-based algorithms, $\text{HUIM-BPSO}_{\text{sig}}$ and HUIM-BPSO, the latter demonstrated

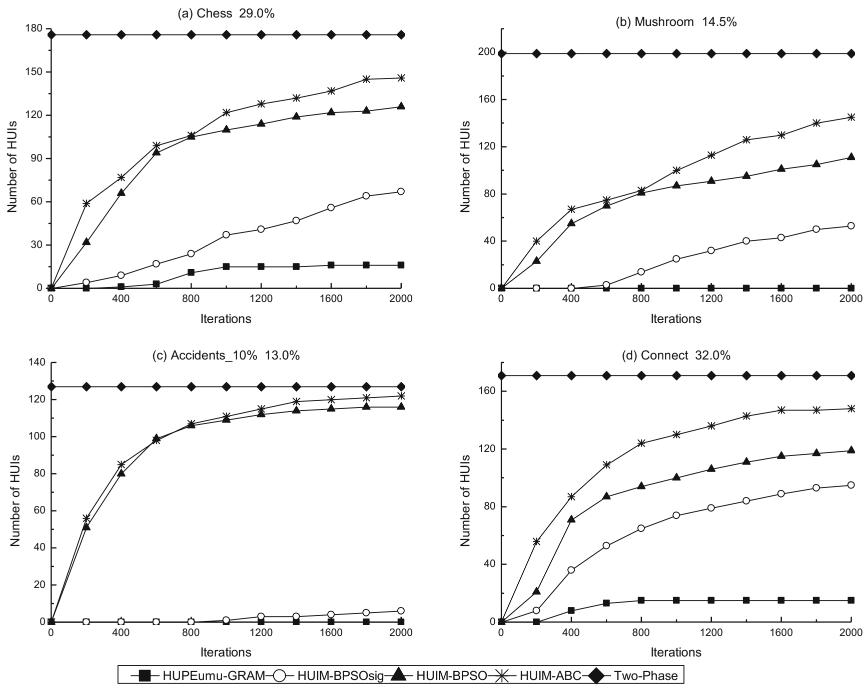

Fig. 3. Convergence performance comparison

better convergence speed because the OR/NOR-tree structure used by HUIM-BPSO avoided invalid combinations of particles. Although HUIM-BPSO demonstrated similar convergence performance to the HUIM-ABC algorithm on Accidents_10%, the HUIM-ABC algorithm always converged faster than HUIM-BPSO. The main reason is that the DNSG strategy of the HUIM-ABC algorithm generated new nectar sources by making use of the discovered HUIs rather than completely at random.

5 Conclusions

In this paper, we proposed an HUI mining algorithm called the HUIM-ABC algorithm based on the ABC algorithm. In the HUIM-ABC algorithm, the problem of HUI discovery was modeled from the perspective of the ABC algorithm. A bitmap was used for information representation. UPBVs could be detected by bitwise operations efficiently. Thus, the useless operation of utility calculation could be avoided. Furthermore, the sizes of discovered HUIs were recorded by different buckets. Based on this, new nectar sources were more likely to be generated within the range of most discovered results' sizes. Thus, more HUIs were mined within limited iteration cycles composed of employed bees, onlooker bees, and scout bees. As a result of the efficient strategies and optimizations introduced, the HUIM-ABC algorithm outperformed existing state-of-the-art HUI mining algorithms based on EC.

Acknowledgments. This work was supported by Beijing Natural Science Foundation (4162022), and High Innovation Program of Beijing (2015000026833ZK04).

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann, San Francisco (1994)
2. Fournier-Viger, P., Lin, J.C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T.: The SPMF open-source data mining library version 2. In: Berendt, B., Bringmann, B., Fromont, É., Garriga, G., Miettinen, P., Tatti, N., Tresp, V. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9853, pp. 36–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46131-1_8
3. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
4. 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)
5. Lin, J.C.-W., Yang, L., Fournier-Viger, J., Wu, M.-T., Hong, T.-P., Wang, S.-L.L., Zhan, J.: Mining high-utility itemsets based on particle swarm optimization. Eng. Appl. Artif. Intell. **55**, 320–330 (2016). https://doi.org/10.1016/j.engappai.2016.07.006
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. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical report, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)
8. Kannimuthu, S., Premalatha, K.: Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl. Artif. Intell. **28**(4), 337–359 (2014). https://doi.org/10.1080/08839514.2014.891839
9. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE Press, New York (1995). https://doi.org/10.1109/icnn.1995.488968
10. Park, J.S., Chen, M.S., Yu, P.S.: An effective hash-based algorithm for mining association rules. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp. 175–186. ACM, New York (1995). https://doi.org/10.1145/223784.223813
11. Song, W., Zhang, Z., Li, J.: A high utility itemset mining algorithm based on subsume index. Knowl. Inf. Syst. **49**(1), 315–340 (2016). https://doi.org/10.1007/s10115-015-0900-1
12. Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. **25**(8), 1772–1786 (2013). https://doi.org/10.1109/TKDE.2012.59