IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 11, 2018, accepted March 20, 2018, date of publication March 26, 2018, date of current version April 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2819162

Mining High Utility Itemsets Using Bio-Inspired Algorithms: A Diverse Optimal Value Framework

WEI SONG[1,2] AND CHAOMIN HUANG[1]

[1]College of Computer Science and Technology, North China University of Technology, Beijing 100144, China
[2]Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, North China University of Technology, Beijing 100144, China

Corresponding author: Wei Song (songwei@ncut.edu.cn)

This work was supported in part by the Beijing Natural Science Foundation under Grant 4162022 and in part by the High Innovation Program of Beijing under Grant 2015000026833ZK04.

ABSTRACT Mining high utility itemsets (HUI) is an interesting research problem in the field of data mining and knowledge discovery. Recently, bio-inspired computing has attracted considerable attention, leading to the development of new algorithms for mining HUIs. These algorithms have shown good performance in terms of efficiency, but are not guaranteed to find all HUIs in a database. That is, the quality is comparatively poor in terms of the number of discovered HUIs. To solve this problem, a new framework based on bio-inspired algorithms is proposed. This approach adjusts the standard roadmap of bio-inspired algorithms by proportionally selecting discovered HUIs as the target values of the next population, rather than maintaining the current optimal values in the next population. Thus, the diversity within populations can be improved. Three new algorithms based on the Bio-HUI framework are developed using the genetic algorithm, particle swarm optimization, and the bat algorithm, respectively. Extensive tests conducted on publicly available datasets show that the proposed algorithms outperform existing state-of-the-art algorithms in terms of efficiency, quality of results, and convergence speed.

INDEX TERMS Data mining, high utility itemset mining, bio-inspired algorithm, genetic algorithm, particle swarm optimization, bat algorithm.

I. INTRODUCTION

Data mining is the non-trivial process of extracting useful and understandable information from different types of vast data repositories [32], [35]. Generally used as the first phase of association rule mining (ARM) [3], [36], frequent itemset mining (FIM) [1], [28] has received considerable attention and been applied in various domains [26], [29]. Given a transaction database, the problem of FIM is to discover sets of items whose occurrence frequency is no less than the minimum support threshold set by users. Generally speaking, frequent itemsets are determined by the frequency of an item occurring in the database. However, different items have different values for specific application scenarios, and items with low occurrence frequencies but high values may not be discovered by traditional FIM algorithms. To solve this problem, utility mining [4], which considers quantity and profit, is emerging as an important research topic.

In utility mining, each item has its own profit and can occur multiple times in one transaction. The utility of an itemset is calculated by summing the product of the item's profit and its occurrence quantity in each relevant transaction. High utility itemsets (HUIs) are those whose utility is no lower than a user-specified threshold. The problem of high utility itemset mining (HUIM) is to discover all HUIs within a transaction database.

Various mining algorithms have been proposed for the discovery of HUIs. Typical algorithms include the level-wise candidate generation-and-test method [18], [23] and techniques based on pattern growth [2], [30]. Some new topics on HUIs are also being discussed, such as the problem of discovering the top-k HUIs [31] and high average-utility itemsets [20]. The existing exact approaches for HUIM tend to degrade as the size of the database and the number of distinct items increase, and the performance may become unacceptable, similar to the problem of FIM applied to social networks or large bioinformatics datasets [5].

To deal with the performance bottleneck of exact approaches, bio-inspired algorithms have been applied for HUIM. For example, the genetic algorithm (GA) has been used to mine HUIs by Kannimuthu and Premalatha [14], and particle swarm optimization (PSO) [21], [22] has recently been applied to the mining of HUIs. These existing HUIM

algorithms based on bio-inspired computing follow the traditional routines of the original GA and PSO algorithm. That is, the optimal values of one population are maintained in the next population. However, HUIM is different from problems in which there are relatively few best values—all itemsets with utilities no lower than the minimum threshold must be discovered. Because the distribution of HUIs is not even, searching with the best values from the previous population as targets may mean that some results are missed within a certain number of iterations.

To solve this problem, we propose a novel bio-inspired-algorithm-based HUIM framework (Bio-HUIF) to discover HUIs. In this framework, rather than choosing only those HUIs with the highest utility values in the current population, roulette wheel selection is applied to all the discovered HUIs to determine the initial target of the next population. Based on Bio-HUIF, three HUIM algorithms are proposed: Bio-HUIF-GA, Bio-HUIF-PSO, and Bio-HUIF-BA. These employ GA, PSO, and the bat algorithm (BA), respectively. For each algorithm, every discovered HUI could be chosen as the initial target of the next population according to the ratio of its utility to the total utilities of all discovered HUIs. The major contributions of this work are summarized as follows.

First, a novel framework for HUIM is proposed based on bio-inspired algorithms. The strategy of selecting discovered HUIs probabilistically, instead of maintaining the best values from population to population, improves the diversity of solutions within a limited number of iterations.

Second, under the proposed framework, three new algorithms are proposed based on GA, PSO, and BA, respectively. Besides the standard concepts of the three bio-inspired algorithms, we use the strategies of bitmap database representation, promising encoding vector checking, and bit difference sets to accelerate the process of HUI discovery.

Third, extensive experiments have been conducted on real datasets to validate the performance of the three algorithms. The results show that the proposed approach outperforms existing bio-inspired HUIM algorithms in terms of efficiency, the number of discovered HUIs, and convergence speed.

The remainder of this paper is organized as follows. In Section II, we describe the problem of HUIM and introduce some related work. The proposed framework is presented in Section III. The three algorithms based on Bio-HUIF are explained in Sections IV–VI, respectively. Experimental results are presented and analyzed in Section VII. Finally, we draw our conclusions in Section VIII.

II. PROBLEM STATEMENT AND RELATED WORK

This section briefly describes the problem of HUIM before reviewing related work in the area of HUIM and bio-inspired algorithms for itemset mining.

A. PROBLEM OF HUIM

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*; 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 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

$$u(X, T_d) = \sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d) \tag{1}$$

The utility of itemset X in $\boldsymbol{D}$ is defined as

$$u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d) \tag{2}$$

The *transaction utility* (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$.

To perform HUIM, 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

$$min\_util = \delta \times \sum_{T_d \in D} TU(T_d) \tag{3}$$

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 [23] is the sum of the transaction utilities of all the transactions containing X, which is defined as

$$TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d) \tag{4}$$

X is a high transaction-weighted utilization itemset (HTWUI) if $TWU(X) \geq min\_util$; otherwise, X is a low transaction-weighted utilization itemset (LTWUI). An HTWUI/LTWUI with k items is called a k-HTWUI/k-LTWUI.

TABLE 1. Example database.

Tid	Transactions	TU
T_1	(*a*, 1) (*c*, 18) (*e*, 1)	27
T_2	(*b*, 6) (*d*, 1) (*e*, 1) (*f*, 1)	66
T_3	(*a*, 2) (*c*, 1) (*e*, 1)	13
T_4	(*d*, 1) (*e*, 1)	11
T_5	(*c*, 4) (*e*, 2)	16
T_6	(*b*, 1) (*f*, 1)	10
T_7	(*b*, 10) (*d*, 1) (*e*, 1)	101
T_8	(*a*, 3) (*c*, 25) (*d*, 3) (*e*, 1)	55
T_9	(*a*, 1) (*b*, 1) (*f*, 3)	15
T_{10}	(*b*, 6) (*c*, 2) (*e*, 2) (*f*, 4)	72

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_1 is $u(e, T_1) = 1 \times 6 = 6$, the utility of itemset ce in transaction T_1 is $u(ce, T_1) = u(c, T_1) + u(e, T_1) = 18 + 6 = 24$, and the utility of itemset ce in the transaction

TABLE 2. Profit table.

Item	a	b	c	d	e	f
Profit	3	9	1	5	6	1

database is $u(ce) = u(ce, T_1) + u(ce, T_3) + u(ce, T_5) + u(ce, T_8) + u(ce, T_{10}) = 24 + 7 + 16 + 31 + 14 = 92$. Given $min\_util = 115$, as $u(ce) < min\_util$, ce is not an HUI. The TU of T_3 is $TU(T_3) = u(ace, T_3) = 13$, and the utilities of other transactions are shown in the third column of Table 1. The TWU of an itemset ce is $TWU(ce) = TU(T_1)+TU(T_3)+TU(T_5) + TU(T_8) + TU(T_{10}) = 183$; thus, ce is an HTWUI.

B. RELATED WORK

1) HUIM ALGORITHMS

As an extension of FIM, HUIM has become an active research problem in data mining. Many algorithms have been proposed for the mining of HUIs.

The basic concepts of HUIM were outlined by Yao *et al.* [34], and the upper bound property was proposed to prune the search space. The Two-Phase algorithm [23] was developed to determine HUIs using the transaction-weighted downward-closure property, similar to the heuristic approach used in FIM [1]. In the Two-Phase algorithm, an HTWUI is used as a superset of the set of HUIs, and only the supersets of the HTWUIs are processed further. Much like the Apriori algorithm for FIM, the main problem of the Two-Phase algorithm is that too many candidates are generated and multiple database scans are required. To reduce the number of candidates in the Two-Phase algorithm, Li *et al.* [18] proposed the strategy of discarding isolated items for HUIM. However, their approach still suffers from the same drawbacks as the candidate generation-and-test scheme for determining HUIs.

To generate candidates efficiently and avoid multiple database scans, the pattern-growth approach [9] and tree-based algorithms have been shown to be efficient for mining HUIs. Examples include IHUP-tree [2], UP-tree [30], and HUI_{TWU}-tree [8]. Although the associated tree structures are often compact, the performance of these methods is closely related to the number of conditional trees constructed, resulting in significant memory requirements.

Other HUI mining algorithms include one based on maximal itemsets [19], the projection-based approach [17], and a bitmap-based method [27].

2) BIO-INSPIRED ALGORITHMS FOR ITEMSET MINING

The field of 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 [15]. Generally speaking, bio-inspired computing optimizes a problem by iteratively improving a candidate solution with regard to a given measure of quality. Biological systems provide abundant inspiration for the construction of high-performance computing models and intelligent algorithms, enabling the production of problem solving techniques with enhanced robustness and flexibility under complex optimization scenarios.

The GA [12] is a typical bio-inspired technique in which each individual has a fitness value that indicates the quality of the solution it represents. Three biologically inspired operators (selection, crossover, and mutation) are applied to give potentially better solutions. There are FIM and ARM algorithms based on GA, such as GAMax [13] and NICGAR [24]. Kannimuthu and Premalatha [14] proposed two GA-based HUIM algorithms, $HUPE_{UMU}$-GARM and $HUPE_{WUMU}$-GARM. The difference between them is that $HUPE_{WUMU}$-GARM does not require the minimum utility threshold. In these algorithms, the selection, crossover, and mutation operators are used iteratively to find HUIs. Because purely random crossover and mutation may produce itemsets that are obviously distinct from the parents, the convergence speed may be low. Thus, both $HUPE_{UMU}$-GARM and $HUPE_{WUMU}$-GARM give only limited results within a certain number of iterations.

PSO [16] is another widely used bio-inspired algorithm. Similar to the GA, PSO is a population-based approach for determining optimal solutions by adopting a velocity to update the particles. Unlike the GA, every particle determines its velocity using the previous velocity, best previous position, and best previous position within its neighborhood. PSO has also been applied to ARM, for example, WARMSWARM [25] and MsP-MmPSO [7]. Lin *et al.* proposed two algorithms for mining HUIs based on PSO, HUIM-BPSOsig [22] and HUIM-BPSO [21]. According to [21], HUIM-BPSO outperforms HUIM-BPSOsig using an OR/NOR-tree structure.

BA [33] is a recently developed bio-inspired algorithm that uses the echolocation behavior of bats to solve optimization problems. Heraguemi *et al.* [10] proposed the BATARM algorithm for ARM . In BATARM, the virtual bat motion models the ARM problem. The same authors later proposed the multi-swarm cooperative bat algorithm MSB-ARM [11]. The algorithm's performance can be improved by applying the ring strategy, master-slave strategy, or a hybrid strategy. To the best of our knowledge, the BA has not been applied to the mining of HUIs.

Generally speaking, most existing HUIM techniques based on bio-inspired mechanisms follow the routine of the standard bio-inspired algorithm. Thus, the search space is further explored according to the optimal values of the former population. This search strategy is suitable for problems with relatively few optimal values. However, for the HUIM problem, the number of results is large. For example, with a minimum utility threshold of 14%, there are 415 HUIs in the Mushroom dataset. With so many results, the standard focusing strategy may miss itemsets that are significantly different from the optimal values of the previous population. One solution to this problem involves enhancing the diversity of each population.

III. THE PROPOSED FRAMEWORK

In this section, we first introduce the bitmap representation of the original database. The pruning strategy of promising encoding vector checking is then proposed. Next, we discuss

the procedure used for the population initialization phase. Finally, the proposed framework is illustrated.

A. BITMAP DATABASE REPRESENTATION

In the proposed framework, the original database is first transformed into a bitmap, which is an effective representation method for mining HUIs [27].

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_M\}$ be a finite set of items and $\boldsymbol{D} = \{T_1, T_2, \ldots, T_N\}$ be a transaction 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 $\boldsymbol{B}(\boldsymbol{D})$ corresponding to transaction T_j $(1 \leq j \leq N)$ and item $i_k (1 \leq k \leq M)$ is denoted (j, k), and is located in the jth row and kth column of $\boldsymbol{B}(\boldsymbol{D})$. The value of (j, k) is defined by

$$B_{j,k} = \begin{cases} 1, & \text{if } i_k \in T_j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

That is, entry (j, k) of $\boldsymbol{B}(\boldsymbol{D})$ is 1 if and only if item i_k is included in transaction T_j; if i_k is not an element of T_j, this entry is set to 0.

In $\boldsymbol{B}(\boldsymbol{D})$, the *bitmap cover* of item i_k, denoted $Bit(i_k)$, is the kth column vector. This naturally extends to itemsets: the bitmap cover of itemset X is defined as $Bit(X) = \text{bitwise-AND}_{i \in X}(Bit(i))$, i.e., it is also a bit vector resulting from the bitwise-AND operation on the bitmap covers of all items included in X. Similarly, for two itemsets X and Y, $Bit(X \cup Y)$ can be computed as $Bit(X) \cap Bit(Y)$, i.e., the bitwise-AND of $Bit(X)$ and $Bit(Y)$.

B. PROMISING ENCODING VECTOR CHECKING

In the proposed framework, an encoding vector is used to represent each individual, i.e., each chromosome, particle, or bat. The encoding vector is composed of 0s or 1s corresponding to whether an item is absent or present in an individual. If the corresponding jth position of an individual contains a 1, the item in the jth position is present in a potential HUI; otherwise, this item is not included and cannot be in a potential HUI. In this paper, the size of the encoding vector representing each individual is equal to the number of 1-HTWUIs in the database.

To speed up the mining process, the concept of promising encoding vectors is defined as follows.

Definition 1: Let *Vec* be an encoding vector composed of 0s or 1s and X be the itemset represented by *Vec*. If $Bit(X)$ is composed only of zeros, *Vec* is called an *unpromising encoding vector* (*UPEV*); otherwise,*Vec* is called a *promising encoding vector* (*PEV*).

Thus, if a newly generated encoding vector is a UPEV, the fitness value computation can be neglected. This technique is called the *PEV check* (PEVC) pruning strategy. The pseudocode of PEVC is shown in Algorithm 1.

Algorithm 1 first calculates the number of 1s in the encoding vector, and identifies which items these 1s represent (Steps 1–2). In Step 3, the result of applying the bitwise-AND operation to all bitmap covers of items in *Vec* is initialized by the bitmap cover of the first item. The main loop

Algorithm 1 Function PEV_Check(*Vec*)

Input Encoding vector *Vec*
Output A PEV of *Vec*

1 Calculate the number of 1s in *Vec*, denoted by *VN*;
2 Denote the *VN* items contained in *Vec* by $i_1, i_2, \ldots, i_{VN}$;
3 $RV = Bit(i_1)$;
4 **for** $k = 2$ to *VN* **do**
5 $RV' = RV \cap Bit(i_k)$;
6 **if** *RV'* is a UPEV **then**
7 $RV' = RV$;
8 Change the bit in *Vec* corresponding to i_k from 1 to 0;
9 **end if**
10 $RV = RV'$;
11 **end for**
12 **Return** *RV*.

(Steps 4–11) executes the PEVC pruning strategy. Step 5 performs the bitwise-AND operation with the bitmap cover of the next item. If the resulting bit vector is a UPEV, this item cannot be included in the final bit vector (Steps 6–9). Step 10 backtracks the result of the bitwise-AND operation. Step 12 returns *RV* for further processing. If *Vec* is a UPEV, executing Algorithm 1 obtains a PEV that is part of *Vec*; otherwise, *Vec* will remain unchanged. In the following algorithms, once a new individual (representing a chromosome, particle, or bat) is generated, the PEVC pruning strategy is executed to ensure that this individual actually occurs in the transaction database.

C. POPULATION INITIALIZATION

In the proposed framework, the initial population is first randomly initialized with *SN* individuals. The initialization process is shown in Algorithm 2.

Algorithm 2 Procedure Pop_Init( )

Input Transaction database $\boldsymbol{D}$, population size *SN*
Output The first population of individuals

1 Scan database $\boldsymbol{D}$ once, and delete 1-LTWUIs;
2 Represent the reorganized database as a bitmap;
3 **for** $i = 1$ to *SN* **do**
4 Generate a random number num_i;
5 Generate a bit vector Vec_i with num_i bits set to 1 using roulette wheel selection;
6 **if** $num_i > 1$ **then**
7 $Vec_i = \text{PEV\_Check}(Vec_i)$;
8 **end if**
9 **end for**

In Algorithm 2, the transaction database is first scanned once to determine the 1-HTWUIs (Step 1). In Step 2, the bitmap representation of the pruned database is constructed. The main loop (Steps 3–9) generates the initial

individuals (chromosomes, particles, or bats) one by one. For each individual, Step 4 assigns a random number of 1s in the ith bit vector, where num_i is an integer between 1 and the number of 1-HTWUIs. Step 5 generates a bit vector with num_i 1s, where the probability that the bit corresponding to i_j will be set to 1 is determined by

$$P_j = \frac{TWU(i_j)}{\sum_{k=1}^{HN} TWU(i_k)} \tag{6}$$

where HN is the number of 1-HTWUIs.

From (6), we can see that if the TWU value of a 1-HTWUI is high, it has a higher probability of being selected in an individual within the first population.

The PEVC pruning strategy described in Algorithm 1 is only performed when $num_i > 1$ (Steps 6–8). This is because each bit in a bit vector corresponds to a 1-HTWUI, so each 1-HTWUI is certainly contained by one or more transactions. Thus, this kind of bit vector is obviously a PEV.

Consider the transaction database in Table 1 and profit table in Table 2. After the first database scan, we obtain the TWU of each item, as presented in Table 3.

TABLE 3. TWU of each item.

Item	a	b	c	d	e	f
TWU	110	264	183	233	361	163

TABLE 4. Reorganized database.

Tid	Transactions	TU
T'_1	(c, 18) (e, 1)	24
T'_2	(b, 6) (d, 1) (e, 1) (f, 1)	66
T'_3	(c, 1) (e, 1)	7
T'_4	(d, 1) (e, 1)	11
T'_5	(c, 4) (e, 2)	16
T'_6	(b, 1) (f, 1)	10
T'_7	(b, 10) (d, 1) (e, 1)	101
T'_8	(c, 25) (d, 3) (e, 1)	46
T'_9	(b, 1) (f, 3)	12
T'_{10}	(b, 6) (c, 2) (e, 2) (f, 4)	72

Given that $min\_util = 115$, as $TWU(a) < min\_util$, item a is deleted. Table 4 lists the reorganized transactions and their TUs for the database in Table 1. In Table 4, a has been removed from transactions T_1, T_3, T_8, and T_9, and the utilities of a have been eliminated from the TUs of these four transactions.

The reorganized database is then represented by a bitmap, as in Table 5.

D. BIO-HUIF

We propose a general framework for HUI mining based on bio-inspired algorithms. A schematic of this framework is illustrated in Fig. 1.

For Bio-HUIF, the first population is initialized by Algorithm 2, and then the iteration number is set to 1.

In any given population, each individual represents an itemset. The fitness value is calculated according to (2),

TABLE 5. Bitmap representation of the example database.

	b	c	d	e	f
T'_1	0	1	0	1	0
T'_2	1	0	1	1	1
T'_3	0	1	0	1	0
T'_4	0	0	1	1	0
T'_5	0	1	0	1	0
T'_6	1	0	0	0	1
T'_7	1	0	1	1	0
T'_8	0	1	1	1	0
T'_9	1	0	0	0	1
T'_{10}	1	1	0	1	1

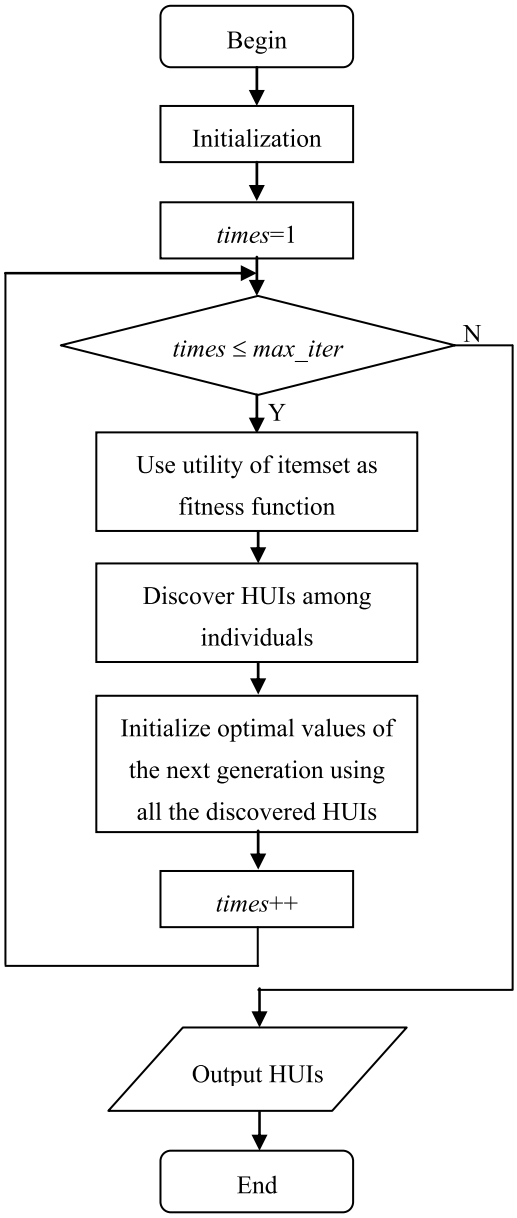


FIGURE 1. Main framework of Bio-HUIF.

and those itemsets with utilities no lower than min_util are kept. The optimal values of the next population are formed by roulette wheel selection among all discovered HUIs.

Namely, the probability that an HUI will be selected as the optimal value of the next population can be obtained from

$$P_i = \frac{fitness_i}{\sum_{j=1}^{|SHUI|} fitness_j} \tag{7}$$

where $SHUI$ is the set of discovered HUIs, $|SHUI|$ denotes the number of elements in $SHUI$, and $fitness_i$ is the fitness value of the ith discovered HUI.

The iteration number is then incremented by one. The above iteration, including fitness value calculation, HUI identification, and next population selection, is repeated until the maximum iteration limit is reached. Finally, all discovered HUIs are output.

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 in average the diversity within one population, and also enhances the efficiency and quality of mining.

According to Bio-HUIF, we propose three algorithms based on GA, PSO, and BA, respectively.

IV. BIO-HUIF-GA

In this section, the bio-inspired GA is briefly described. The proposed Bio-HUIF-GA is then presented in detail. Finally, an example is used to illustrate the application of Bio-HUIF-GA.

A. BASIC IDEA OF GA

The GA is possibly the first bio-inspired algorithm to simulate genetic systems for complex optimization problems [12]. GAs operate on a population of chromosomes, each of which is a potential solution to a given problem. Only those individuals in a population who are better suited to the environment are likely to survive and generate offspring.

In the GA, the first population is generated randomly or heuristically. Then, the three typical operators of selection, crossover, and mutation are applied iteratively to generate new populations. Selection is responsible for determining which individuals produce offspring using the predefined fitness function. Crossover combines parts of two parent chromosomes to produce child chromosomes for the next population. Mutation maintains diversity in the population according to defined probabilities and inhibits premature convergence. The process of selection, crossover, and mutation is repeated until some termination condition is satisfied.

B. ALGORITHM DESCRIPTION

Based on Bio-HUIF, Algorithm 3 describes the proposed Bio-HUIF-GA for mining HUIs.

Algorithm 3 Bio-HUIF-GA

Input Transaction database $\boldsymbol{D}$, minimum utility value min_util, maximum number of iterations max_iter

Output HUIs

```
1   Pop_Init( );
2   times=1;
3   SHUI = ∅;
4   while times < max_iter do
5    for each chromosome C_i do
6     X = IS(C_i);
7     if u(X) ≥ min_util & X ∉ SHUI then
8      X → SHUI;
9     end if
10    end for
11    Next_Gen_GA( );
12    Select two HUIs h_i and h_j from SHUI using
      roulette wheel selection;
13    Represent h_i and h_j as bit vectors CN_i and CN_j;
14    Replace two randomly selected chromosomes
      of the current population by CN_i and CN_j.
15     times++;
16   end while
17   Output all HUIs.
```

Algorithm 3 describes the procedure Pop_Init is called in Step 1. Step 2 then initializes the iteration number to 1. In Step 3, the set of all HUIs $SHUI$ is initialized to the empty set. The main loop (Steps 4–16) discovers HUIs population by population. Each chromosome of the population is checked in the loop from Steps 5–10. Step 6 determines the itemset that corresponds to the enumerating chromosome. Here, the function $IS(\,)$ gives itemset X by unifying the items in C_i if its value is 1. If the current chromosome can produce an HUI X that has not already been discovered (Step 7), Step 8 records this itemset. Step 11 calls the procedure Next_Gen_GA to generate the next population (described in Algorithm 4). In Steps 12–14, two discovered HUIs are selected using (7) and represented as bit vectors. The two selected bit vectors are then used to replace the two randomly selected chromosomes in the new population. Thus, the diversity of the new population is improved. Step 15 updates the iteration number. Finally, all discovered HUIs are output in Step 17.

In Algorithm 4, the number of chromosomes in the new population, SN_{new}, is initialized to 0 in Step 1. The main loop from Steps 2–12 produces the next population based on the current SN chromosomes. Step 3 selects two chromosomes by roulette wheel selection. That is, individuals with higher utilities will have a higher probability of being selected. In Step 4, the different bits of the two selected chromosomes are recorded by $BitDiff$, which is defined as follows.

Definition 2: Let Vec_i and Vec_j be two bit vectors with len bits. The *bit difference set* is $BitDiff\,(Vec_i, Vec_j) = \{num | 1 \leq num \leq len, b_{num}\,(Vec_i) \oplus b_{num}(Vec_j) = 1\}$, where $b_{num}\,(Vec_i)$ is the num-th bit of Vec_i and $\oplus$ denotes exclusive disjunction.

Step 5 then calculates the number of crossover bits in the two selected chromosomes by:

$$cnum = \lfloor |BitDiff\,(C_i, C_j)| r \rfloor \tag{8}$$

Algorithm 4 Procedure Next_Gen_GA( )

Input The current population
Output The next population

1 $SN_{new} = 0$;
2 **while** $SN_{new} < SN$ **do**
3 Select two chromosomes C_i and C_j from SN chromosomes using (7);
4 Record the bits in which C_i is different from C_j as *BitDiff*;
5 Calculate *cnum* using (8);
6 **for** $k = i$ to j **do**
7 Randomly select *cnum* bits from elements of *BitDiff*, and change the selected bits of C_k from 0 to 1 or from 1 to 0;
8 Randomly change one bit of C_k from 0 to 1 or from 1 to 0;
9 C_k = PEV_Check(C_k);
10 $SN_{new}++$;
11 **end for**
12 **end while**

where r is a random number in the range (0, 1), $|BitDiff(C_i, C_j)|$ is the number of elements in $BitDiff(C_i, C_j)$, and $\lfloor |BitDiff(C_i, C_j)|r \rfloor$ denotes the largest integer that is less than or equal to $|BitDiff(C_i, C_j)|r$.

The loop from Steps 6–11 generates two new chromosomes of the next population using the idea of GA. For each chromosome, Steps 7 and 8 simulate the crossover and mutation of GA, respectively. That is, *cnum* bits are selected at random from *BitDiff* and changed using the bitwise complement operation (Step 7). Then, one bit is randomly selected and changed from 0 to 1 or from 1 to 0 (Step 8). The PEVC pruning strategy described in Algorithm 1 is called in Step 9. In Step 10, the number of chromosomes in the new population is incremented by one.

C. ILLUSTRATED EXAMPLE

We use the transaction database in Table 1 and profit table in Table 2 for explanation. We also suppose that $min\_util = 115$, and assume that the 1-LTWUIs have been deleted and the database has been transformed to the form in Table 5. Assume the size of population *SN* to be 3. As the number of 1-HTWUIs is 5, there are five bits in the bit vector for chromosome encoding. First, *SHUI* is initialized as the empty set. To generate the first chromosome, a random number (in this case, 4) is first generated to indicate the number of 1s in the first chromosome. To determine which bits are set to 1, (6) is used. Assume the bit vector of the first chromosome is "11110". We obtain the other two chromosomes using the same method; the three chromosomes of the first population are shown in Fig. 2.

We can see that the first chromosome C_1 represents itemset *bcde*. According to Algorithm 1, *RV* is initialized by *Bit*(*b*). Then, $RV \cap Bit(c) = 0100011011 \cap 1010100101 = 0000000001$. As this result is a PEV, *RV* is updated

	b	*c*	*d*	*e*	*f*
C_1	1	1	1	1	0
C_2	1	0	1	1	0
C_3	0	1	0	1	0

FIGURE 2. Initial chromosomes.

to 0000000001. Next, $RV \cap Bit(d) = 0000000000$, so item *d* is deleted from C_1, and *RV* retains the value 0000000001. Then, $RV \cap Bit(e) = 0000000001$; as this is a PEV, the final value of *RV* is 0000000001. Thus, C_1 is 11010 representing itemset *bce*, and this itemset is contained in T'_{10}. Because $u(bce) = 68 < min\_util$, *bce* is not an HUI.

Similarly, C_2 is a PEV representing itemset *bde*. As *bde* is contained in T'_2 and T'_7, and $u(bde) = 166 > min\_util$, $SHUI = \{bde : 166\}$, where the number after the colon denotes the utility. Furthermore, C_3, representing itemset *ce*, is not an HUI, and so *SHUI* remains unchanged. At this point, the first population is composed of the three chromosomes shown in Fig. 3.

	b	*c*	*d*	*e*	*f*
C_1	1	1	0	1	0
C_2	1	0	1	1	0
C_3	0	1	0	1	0

FIGURE 3. Chromosomes of the first population.

Suppose C_1 and C_2 are selected at first. As $11010 \oplus 10110 = 01100$, $BitDiff(C_1, C_2) = \{2, 3\}$, that is, C_1 and C_2 differ in their second and third bits. Suppose the random number r is 0.5, $cnum = \lfloor 2 * 0.5 \rfloor = 1$. For C_1, one bit (either the second or third) will be selected for crossover. Suppose the second bit is selected to be changed from 1 to 0. Then, the new C_1 is 10010. Next, suppose the fifth bit is randomly selected for mutation. After changing the fifth bit from 0 to 1, C_1 becomes 10011, representing itemset *bef*. This is a PEV, contained by transactions T'_2 and T'_{10}, and $u(bef) = 131 > min\_util$, $SHUI = \{bde: 166, bef: 131\}$. We can also obtain the child of C_2 using the same routine. Suppose the new C_2 is 10000, representing itemset *b*. As $u(b) = 216 > min\_util$, $SHUI = \{bde: 166, bef: 131, b: 216\}$. After generating these two new chromosomes, $SN_{new} = 2$. As $SN_{new} < SN$, two chromosomes of the current population will be selected and the above process of crossover and mutation will repeat until $SN_{new} \geq SN$.

Suppose that no HUIs are generated by the other child chromosomes of C_3 and C_4, and *SHUI* is still $\{bde: 166, bef: 131, b: 216\}$. According to Algorithm 3, two HUIs will be

selected using (7). That is, b has the highest probability of being selected and bef has the lowest probability of being selected. After representing the two selected HUIs as bit vectors CN_i and CN_j, two randomly selected chromosomes from the second population will be replaced by CN_i and CN_j. The above process is then repeated for the new population until the termination condition is satisfied.

V. BIO-HUIF-PSO

In this section, the PSO algorithm is briefly discussed before the proposed Bio-HUIF-PSO algorithm is explained. Finally, an example is used to illustrate the application of Bio-HUIF-PSO.

A. BASIC IDEA OF PSO

PSO is a bio-inspired algorithm that simulates the foraging behavior of birds or fish [16].

In the PSO algorithm, several particles are initialized at random. Each particle moves toward the optimal value according to the following two equations:

$$v_i^{t+1} = wv_i^t + c_1r_1(pbest_i - x_i^t) + c_2r_2(gbest - x_i^t) \quad (9)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (10)$$

where v_i^t and v_i^{t+1} are the velocities of the ith particle at iterations t and $t+1$, x_i^t and x_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).

The first part of (9) includes the previous velocity and constitutes the momentum component; the second part uses the previous best position and constitutes the cognitive component; the third part takes the best previous position of the neighborhood and constitutes the social component of the iteration.

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

B. ALGORITHM DESCRIPTION

Based on Bio-HUIF, Algorithm 5 describes the proposed Bio-HUIF-PSO algorithm for mining HUIs.

In Algorithm 5, the procedure Pop_Init is called in Step 1. Step 2 then initializes the iteration number to 1. In Step 3, the global best particle $gbest$ is initialized as the empty set. The set of all HUIs $SHUI$ is also initialized as the empty set in Step 4. The main loop (Steps 5–22) discovers HUIs population by population. The SN particles are checked one by one in the loop from Steps 6–17. If the current population is the first population, the current particle P_i is also initialized as $pbest_i$ (Steps 7–9). Step 10 determines the itemset that corresponds 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 HUI X that has not already been discovered (Step 11), Step 12 records this itemset. Steps 14–16 update the $pbest$ of the current particle. In Step 18, $gbest$ is updated by the particle corresponding to the discovered HUI with the highest utility value. Using the new $gbest$, Step 19 calls the procedure Next_Gen_PA to generate the next population (described in Algorithm 6). In Step 20, initial $gbest$ of the next population is selected by (7) using all discovered HUIs. Step 21 updates the iteration number. Finally, all discovered HUIs are output in Step 23.

Algorithm 5 Bio-HUIF-PSO

Input Transaction database $\boldsymbol{D}$, minimum utility value min_util, maximum number of iterations max_iter
Output HUIs

```
1   Pop_Init( );
2   times=1;
3   gbest = ∅;
4   SHUI = ∅;
5   while times < max_iter do
6    for i = 1 to SN do
7     if times==1 then
8      pbest_i = P_i;
9     end if
10   X = IS(P_i);
11   if u(X) ≥ min_util and X ∉ SHUI then
12     X → SHUI;
13   end if
14   if u(X) > u(pbest_i) then
15     pbest_i = P_i;
16   end if
17   end for
18   Find gbest among SN particles;
19   Next_Gen_PA( );
20   Determine gbest using roulette wheel selection
     among HUIs in SHUI;
21   times++;
22   end while
23   Output all HUIs.
```

In Algorithm 6, the main loop in Steps 1–8 produces the next population of SN particles. For the problem of HUIM, the velocity of standard PSO calculated by (9) is adapted to an integer, which indicates how many bits are to be changed for a particle. Thus, we rewrite (9) using

$$v_i = v_{i1} + v_{i2} + v_{i3} \quad (11)$$

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

$$v_{i2} = \lfloor |BitDiff(P_i, pbest_i)| r_1 \rfloor \quad (12)$$

$$v_{i3} = \lfloor |BitDiff(P_i, gbest)| r_2 \rfloor \quad (13)$$

In (11), v_{i1} approaches the optimal values at random, v_{i2} approaches the optimal values using the difference from the best previous position of P_i, and v_{i3} approaches the optimal values using the difference from the best result of the current population. Considering the problem of HUIM, the bit

Algorithm 6 Procedure Next_Gen_PA( )

Input The current population
Output *SN* particles of the next population

1 **for** each particle P_i **do**
2 Randomly change one bit of P_i from 0 to 1 or from 1 to 0;
3 Calculate v_{i2} of P_i using (12);
4 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;
5 Calculate v_{i3} of P_i using (13);
6 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;
7 P_i = PEV_Check(P_i);
8 **end for**

difference set (see Definition 2) is used to model the difference between two bit vectors.

Thus, in Steps 2–6, 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. Similar to Algorithm 4, both v_{i2} bits and v_{i3} bits are selected at random from elements of *BitDiff*. The PEVC pruning strategy described in Algorithm 1 is called in Step 7, ensuring that the generated particle is included in one or more transactions.

C. ILLUSTRATED EXAMPLE

Using the same initial method as the example described in Section IV, the first population is $P_1 = 11010$, $P_2 = 10110$, $P_3 = 01010$.

This example differs from that in Section IV in two aspects. First, there is a set representing *pbest* for each particle. For the first population, $pbest_i$ is the same as P_i. Thus, $pbest_1 = 11010$, $pbest_2 = 10110$, $pbest_3 = 01010$. Second, the bit vector representing the HUI with the highest utility is stored as *gbest*. Here, *gbest* is 10110, and *SHUI*= {*bde*: 166} at this point.

As an example of how the next population is generated, let us consider P_1. First, suppose the 5th bit to have been selected at random, so this bit is changed from 0 to 1. P_1 is now 11011. Assuming the random number r_1 is 0.5, we have $BitDiff(P_1, pbest_1) = \{5\}$, so v_{12} is $\lfloor 1 * 0.5 \rfloor = 0$. Thus, P_1 remains 11011. Next, suppose r_2 is randomly generated as 0.8. As $BitDiff(P_1, gbest) = \{2, 3, 5\}$, v_{13} is $\lfloor 3 * 0.8 \rfloor = 2$. Thus, we randomly change two bits of P_1 in *BitDiff* using the bitwise complement operation. Suppose the 2nd and 3rd bits are selected; the 2nd bit is changed from 1 to 0, and the 3rd bit is changed from 0 to 1. Thus, the new P_1 is 10111, representing itemset *bdef*. We can see that P_1 is a PEV, and $u(bdef) = 66 < min\_util$. Thus, both *SHUI* and *gbest* are unchanged. Because $u(bdef) < u(bce)$, $pbest_1$ is also unchanged.

Similarly, we can obtain new values for P_2 and P_3 with the same roadmap. After the second iteration, suppose *SHUI* = {*bde*: 166, *be*: 222}, $pbest_1 = 11010$, $pbest_2 = 10110$, $pbest_3 = 10010$, and $gbest = 10010$. Using the standard PSO algorithm, 10010 (the bit vector representing HUI *be*) will certainly be the initial *gbest* of the next population. However, in the Bio-HUIF-PSO algorithm, 10110 representing *bde*, which is the discovered HUI with lower utility, may be selected as the global best value for the next iteration with a probability of $166/(166 + 222) \approx 0.428$.

The above process is repeated until the maximum number of iterations is reached.

VI. BIO-HUIF-BA

In this section, the BA method is briefly introduced and the proposed Bio-HUIF-BA is described in detail. An example is presented to illustrate the application of Bio-HUIF-BA.

A. BASIC IDEA OF BA

The newly developed BA is based on the echolocation behavior of bats, which vary the pulse rates and loudness of emissions when searching for prey and avoiding obstacles [33].

In the BA, each bat is randomly generated and approaches the optimal solution by changing its frequency, velocity, and position. In each population, these values are updated by

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{14}$$
$$v_i^{t+1} = v_i^t + (x_i^t - gbest)f_i \tag{15}$$
$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{16}$$

where f_i is the frequency of the ith bat (used to adjust the velocity), f_{min}, f_{max} are the minimum/maximum frequencies of the pulses emitted by all bats, $\beta \in [0, 1]$ is a random number, v_i^t, v_i^{t+1} are the velocities of the ith bat at iterations t and $t + 1$, x_i^t, x_i^{t+1} are the locations of the ith bat at iterations t and $t + 1$, and *gbest* is the current global best location.

When approaching prey, bat B_i will decrease its loudness and increase the rate of pulse emission. This phenomenon can be simulated by the following equations:

$$A_i^{t+1} = \alpha A_i^t \tag{17}$$
$$r_i^{t+1} = r_i^0(1 - \exp(-\gamma t)) \tag{18}$$

where A_i^t, A_i^{t+1} denote the loudness at iterations t and $t + 1$, r_i^0 is the initial rate of pulse emission, r_i^{t+1} is the rate of pulse emission at iteration $t + 1$, and $0 < \alpha < 1$, $\gamma > 0$ are constants.

All bats update their velocities, locations, loudness, and pulse emission rates repeatedly, until the best solution is found or the maximum number of iterations is reached.

B. ALGORITHM DESCRIPTION

Based on Bio-HUIF, Algorithm 7 describes the proposed Bio-HUIF-BA for mining HUIs.

In Algorithm 7, the procedure Pop_Init is called in Step 1. Step 2 then initializes the iteration number to 1. In Step 3, the

Algorithm 7 Bio-HUIF-BA

Input Transaction database $\boldsymbol{D}$, minimum utility value *min_util*, maximum number of iterations *max_iter*
Output HUIs

1 Pop_Init( );
2 *times*=1;
3 $gbest = \emptyset$;
4 $SHUI = \emptyset$;
5 **while** *times* < *max_iter* **do**
6 **for** $i = 1$ to *SN* **do**
7 **if** *times*==1 **then**
8 Initialize A_i and r_i;
9 **end if**
10 $X = IS(B_i)$;
11 **if** $u(X) \geq min\_util \& X \notin SHUI$ **then**
12 $X \rightarrow SHUI$;
13 **end if**
14 **end for**
15 Find *gbest* among *SN* bats;
16 Next_Gen_BA( );
17 Determine *gbest* using roulette wheel selection among HUIs in *SHUI*;
18 *times*++;
19 **end while**
20 Output all HUIs.

Algorithm 8 Procedure Next_Gen_BA( )

Input The current population
Output *SN* bats of the next population

1 **for** each bat B_i **do**
2 Randomly change one bit of B_i from 0 to 1 or from 1 to 0;
3 Calculate its frequency using (14);
4 Calculate v_{i2} of B_i using (20);
5 Randomly select v_{i2} bits from elements of *BitDiff*, and change v_{i2} bits of B_i from 0 to 1 or from 1 to 0;
6 B_i = PEV_Check(B_i);
7 **if** $rand_1 > r_i$ **then**
8 $X = IS(B_i)$;
9 **if** $u(X) \geq min\_util \ \& \ X \notin SHUI$ **then**
10 $X \rightarrow SHUI$;
11 **end if**
12 Randomly change one bit of B_i using bitwise complement operation;
13 B_i = PEV_Check(B_i);
14 **end if**
15 **if** $rand_2 < A_i \ \& \ u(X) < u(IS(gbest))$ **then**
16 Reduce A_i using (17);
17 Increase r_i using (18);
18 **end if**
19 **end for**

global best bat *gbest* is initialized as the empty set. The set of all HUIs *SHUI* is also initialized as the empty set in Step 4. The main loop (Steps 5–19) discovers HUIs population by population. The *SN* bats are each checked in the loop from Steps 6–14. If the current population is the first population, both *A* and *r* of the current bat are initialized. Step 10 determines the itemset that corresponds to the enumerating bat. Here, the function *IS*( ) is the same as in Algorithms 3 and 5. If the current bat can produce an HUI *X* that has not already been discovered (Step 11), Step 12 records this itemset. In Step 15, *gbest* is updated by the bat corresponding to the discovered HUI with the highest utility value. Using the new *gbest*, Step 16 calls the procedure Next_Gen_BA to generate the next population (described in Algorithm 8). In Step 17, the initial *gbest* of the next population is selected by (7) using all discovered HUIs. Step 18 updates the number of iterations. Finally, all discovered HUIs are output in Step 20.

In Algorithm 8, the main loop from Steps 1–19 produces the next population of *SN* bats. Similar to Algorithm 6, we rewrite (15) for the HUIM problem as

$$v_i = v_{i1} + v_{i2} \tag{19}$$

where v_{i1} is always set to 1 and v_{i2} is calculated by

$$v_{i2} = \lfloor |BitDiff(B_i, gbest)| f_i \rfloor \tag{20}$$

In (19), v_{i1} approaches the optimal values at random, whereas v_{i2} approaches the optimal values using the difference between the current bat and the best result of the current population. Thus, in Steps 2–5, bat B_i randomly changes one bit using the bitwise complement operation, and then randomly changes v_{i2} bits from elements of *BitDiff* from 0 to 1 or from 1 to 0. The PEVC pruning strategy described in Algorithm 1 is called in Step 6.

Using the idea of BA, if a random number $rand_1$ is greater than r_i, that is, the rate of pulse emission of the current B_i (Step 7), Steps 8–11 determine the itemset that corresponds to the enumerating bat and records it if it has not already been discovered. A new B_i is then generated by randomly changing one bit of the original bat from 0 to 1 or from 1 to 0 (Step 12). The PEVC pruning strategy described in Algorithm 1 is called in Step 13. Steps 15–18 update A_i and r_i using the standard idea of BA.

C. ILLUSTRATED EXAMPLE

Using the same initial method as the example described in Section IV, the first population is $B_1 = 11010$, $B_2 = 10110$, $B_3 = 01010$.

The main difference between this example and that in Section V is that *pbest* is not considered here. Thus, $gbest = 10110$ and $SHUI = \{bde : 166\}$ at this point.

As an example of the generation of the next population, let us consider B_1. First, suppose the 5th bit is selected at random, so this bit is changed from 0 to 1. B_1 is now 11011. Given $f_{min} = 0$, $f_{max} = 1$, suppose the randomly generated β is 0.8; then, $f_1 = 0.8$ according to (14). Next, we compute the

velocity of B_1. We have $BitDiff(B_1, gbest) = \{2, 3, 5\}$, and $v_{12} = \lfloor 3 * 0.8 \rfloor = 2$. Thus, we randomly change two bits of B_1 using the bitwise complement operation. Suppose the 2nd and 3rd bits are selected; the 2nd bit is changed from 1 to 0, and the 3rd bit is changed from 0 to 1. Thus, the new B_1 is 10111. A random number $rand_1$ is then generated. For the sake of explanation, we suppose $rand_1$ is greater than the current r_1. The current B_1 is 10111, representing itemset *bdef*. We can see that B_1 is a PEV, and $u(bdef) = 66 < min\_util$. Thus, both *SHUI* and *gbest* are unchanged. Then, B_1 is further processed by randomly changing one bit using the bitwise complement operation. Suppose the 3rd bit is changed from 1 to 0, so that B_1 is now 10011. We can see that B_1 is a PEV that represents itemset *bef*. As $u(bef) = 131 > min\_util$, $SHUI = \{bde : 166, bef:131\}$. As $u(bef) < u(bde)$, *gbest* is unchanged.

Similarly, we can obtain new values for B_2 and B_3 with the same roadmap. From all of the HUIs in *SHUI*, the *gbest* of the next population is then determined by (7). That is, each discovered HUI may be selected as the initial *gbest* of the next iteration, and the selection probability is proportional to each HUI's utility. The above process is repeated until the maximum number of iteration is reached.

VII. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our algorithms and compare them with two bio-inspired HUIM algorithms, $HUPE_{UMU}$-GARM [14] and HUIM-BPSO [21]. Furthermore, this set of comparisons includes two exact HUIM algorithms, IHUP [2] and UP-Growth [30]. The source code of each algorithm was downloaded from the SPMF data mining library [6].

A. 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 real datasets were used to evaluate the performance of the algorithms. The characteristics of the datasets are presented in Table 6.

TABLE 6. Characteristics of the datasets.

Dataset	Avg.Trans.Len.	#Items	#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 [6]. The Chess dataset originates from game steps. The Mushroom dataset contains various species of mushrooms and their characteristics, such as shape, odor, and habitat. The Accident dataset is composed of (anonymized) traffic accident data. Similar to the work of Lin *et al.* [21], only 10% of the total dataset was used for experiments. The Connect dataset is also derived from game steps.

For all experiments, the termination criterion was set to 2,000 iterations and the initial population size was set to 20.

B. RUNTIME

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

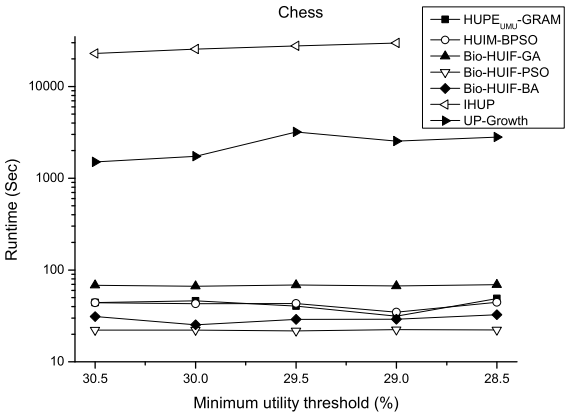


FIGURE 4. Execution times for the Chess dataset.

Fig. 4 compares the execution times for the Chess dataset. We can see that Bio-HUIF-PSO and Bio-HUIF-BA are faster than the other algorithms, with Bio-HUIF-PSO achieving the best performance. $HUPE_{UMU}$-GARM and HUIM-BPSO show similar runtimes on this dataset. Although the proposed Bio-HUIF-GA is slower than $HUPE_{UMU}$-GARM and HUIM-BPSO, it is always faster than the two exact algorithms, IHUP and UP-Growth. When the minimum utility threshold is 28.5%, IHUP cannot return any results because of the excessive memory requirements. On average, both Bio-HUIF-GA and Bio-HUIF-BA are one order of magnitude faster than UP-Growth, and Bio-HUIF-PSO is two orders of magnitude faster than UP-Growth.

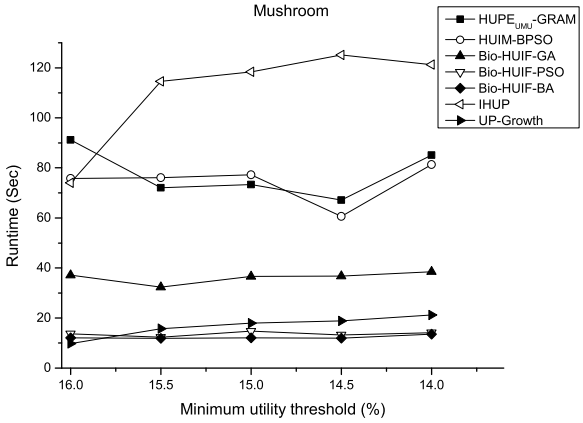


FIGURE 5. Execution times for the Mushroom dataset.

With the Mushroom dataset (see Fig. 5), Bio-HUIF-BA achieves the best performance, being slightly faster than Bio-HUIF-PSO. All three of the proposed algorithms are more

efficient than the other two bio-inspired HUIM algorithms, $HUPE_{UMU}$-GARM and HUIM-BPSO. For example, Bio-HUIF-BA is 5.33 times faster than $HUPE_{UMU}$-GARM and 5.04 times faster than HUIM-BPSO on average. Regarding the two exact algorithms, UP-Growth is slower than Bio-HUIF-BA and Bio-HUIF-PSO and faster than the other algorithms, whereas IHUP is still slower than all the other algorithms on average.

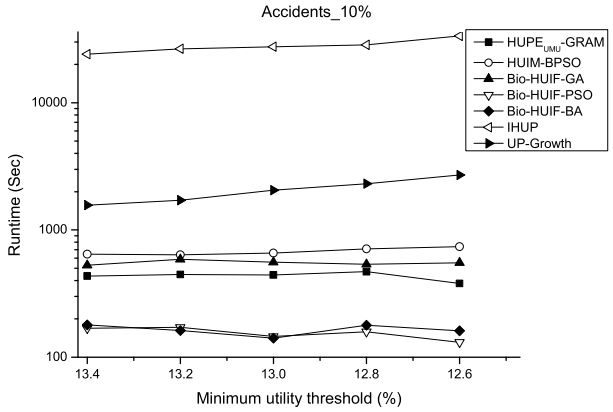


FIGURE 6. Execution times for the Accidents_10% dataset.

From Fig. 6, we can see that all three proposed algorithms are faster than the two exact algorithms when applied to the Accidents_10% dataset. For example, Bio-HUIF-GA is one order of magnitude faster than IHUP, and both Bio-HUIF-PSO and Bio-HUIF-BA are two orders of magnitude faster than IHUP. Being slightly faster than Bio-HUIF-BA, Bio-HUIF-PSO gives the best runtime performance on this dataset. The performance of Bio-HUIF-GA is between that of $HUPE_{UMU}$-GARM and HUIM-BPSO.

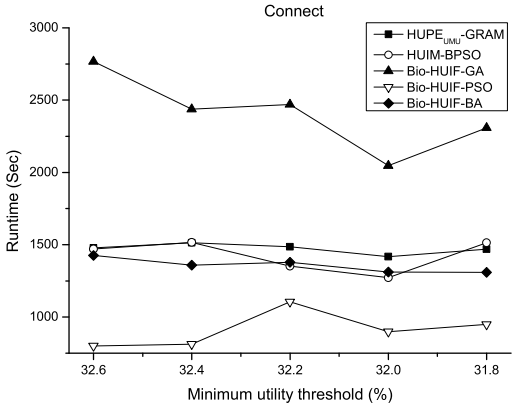


FIGURE 7. Execution times for the Connect dataset.

For the Connect dataset, when the minimum utility threshold is between 31.8% and 32.6%, both IHUP and UP-Growth run out of memory. Thus, in Fig. 7, the two exact algorithms are not plotted. For the five bio-inspired algorithms, Bio-HUIF-PSO is obviously more efficient than the other four algorithms. For example, Bio-HUIF-PSO is 8.59 times faster than $HUPE_{UMU}$-GARM and 8.28 times faster than HUIM-BPSO. The proposed Bio-HUIF-BA is also faster than $HUPE_{UMU}$-GARM and HUIM-BPSO. Bio-HUIF-GA is always slower than the other four algorithms.

C. 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 percentage of discovered HUIs among the five bio-inspired algorithms. As the two exact algorithms, IHUP and UP-Growth, can certainly find all HUIs, we do not illustrate their results. The comparison results are presented in Tables 7–10.

TABLE 7. Percentage (%) of discovered HUIs for the Chess dataset.

Minimum utility threshold	$HUPE_{UMU}$-GARM	HUIM-BPSO	Bio-HUIF-GA	Bio-HUIF-PSO	Bio-HUIF-BA
28.5	7.87	66.23	100	97.05	96.39
29.0	11.36	60.80	100	99.43	97.73
29.5	11.71	69.37	100	100	100
30.0	20.59	82.35	100	100	100
30.5	23.53	91.18	100	100	100
Average	15.01	73.98	100	99.30	98.82

TABLE 8. Percentage (%) of discovered HUIs for the Mushroom dataset.

Minimum utility threshold	$HUPE_{UMU}$-GARM	HUIM-BPSO	Bio-HUIF-GA	Bio-HUIF-PSO	Bio-HUIF-BA
14.0	4.10	43.37	100	99.04	75.90
14.5	13.07	58.79	100	97.99	88.44
15.0	0	68.18	100	100	98.86
15.5	27.03	81.08	100	100	100
16.0	38.10	95.24	100	100	100
Average	16.46	69.33	100	99.41	92.64

TABLE 9. Percentage (%) of discovered HUIs for the Accidents_10% dataset.

Minimum utility threshold	$HUPE_{UMU}$-GARM	HUIM-BPSO	Bio-HUIF-GA	Bio-HUIF-PSO	Bio-HUIF-BA
12.6	7.14	85.71	100	97.62	95.24
12.8	6.96	88.61	100	99.37	95.57
13.0	12.60	90.55	100	100	99.21
13.2	24.21	96.84	100	100	98.95
13.4	18.06	94.44	100	100	100
Average	13.79	91.23	100	99.40	97.79

TABLE 10. Percentage (%) of discovered HUIs for the Connect dataset.

Minimum utility threshold	$HUPE_{UMU}$-GARM	HUIM-BPSO	Bio-HUIF-GA	Bio-HUIF-PSO	Bio-HUIF-BA
31.8	8.43	57.09	100	99.23	99.62
32.0	13.45	60.23	100	100	99.41
32.2	13.27	79.59	100	100	100
32.4	19.40	94.03	100	100	100
32.6	39.29	82.14	100	100	100
Average	18.77	74.62	100	99.85	99.81

We can see that all three of the proposed algorithms outperform the other two bio-inspired algorithms in terms of number of discovered HUIs. Specifically, Bio-HUIF-GA, Bio-HUIF-PSO, and Bio-HUIF-BA can discover more than 90% of all the HUIs, with the lowest average percentage being the 92.64% given by Bio-HUIF-BA for the Mushroom dataset. Note that Bio-HUIF-GA can find all HUIs under all listed thresholds for all four datasets. On the contrary, $HUPE_{UMU}$-GARM, the other HUIM algorithm based on GA, cannot discover more than 20% of all HUIs with any of the four datasets. Similarly, the Bio-HUIF-PSO algorithm outperforms the other PSO-based HUIM-BPSO algorithm in terms of discovered HUIs. For example, Bio-HUIF-PSO discovers an average of 30% more HUIs with the Mushroom dataset. This set of experiments shows that the proposed Bio-HUIF improves the population diversity of bio-inspired HUIM algorithms.

D. CONVERGENCE

In this section, the convergence is evaluated for all the datasets. Since the two exact algorithms IHUP and UP-Growth can discover all HUIs, we only plot the convergence performance of the five bio-inspired HUIM algorithms. The results w.r.t. different number of iterations are shown in Figs. 8–11.

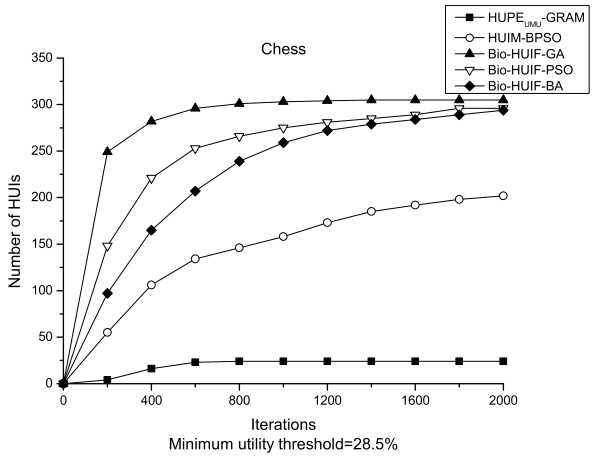


FIGURE 8. Convergence performance comparison for the Chess dataset.

For this set of experiments, we can observe that the convergence speed of $HUPE_{UMU}$-GARM was lower than that of the all other algorithms. This is because the standard GA-based algorithm suffered from the combination explosion problem in the evolution process composed of selection, crossover, and mutation.

Although HUIM-BPSO demonstrated similar convergence performance to the Bio-HUIF-BA algorithm on Accidents_10% when the number of iterations is small, the Bio-HUIF-BA algorithm converges faster than HUIM-BPSO when the number of iterations is higher than 600.

For all the other cases, all the three proposed algorithms converge more efficient than the other two bio-inspired HUIM algorithms, $HUPE_{UMU}$-GARM and HUIM-BPSO.

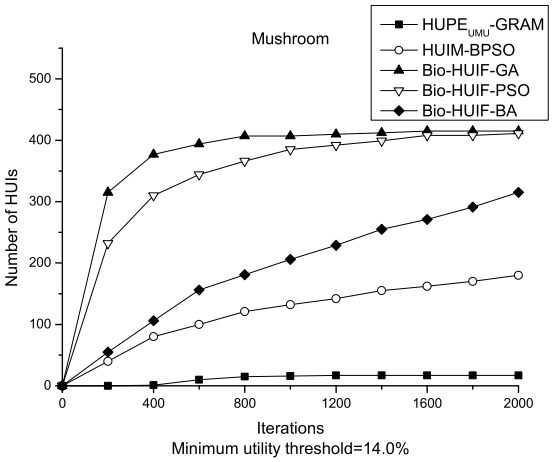


FIGURE 9. Convergence performance comparison for the Mushroom dataset.

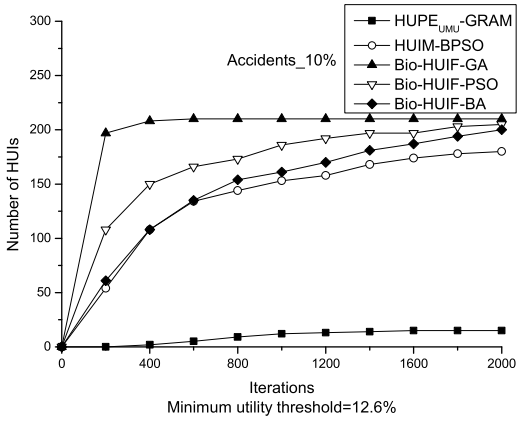


FIGURE 10. Convergence performance comparison for the Accidents_10% dataset.

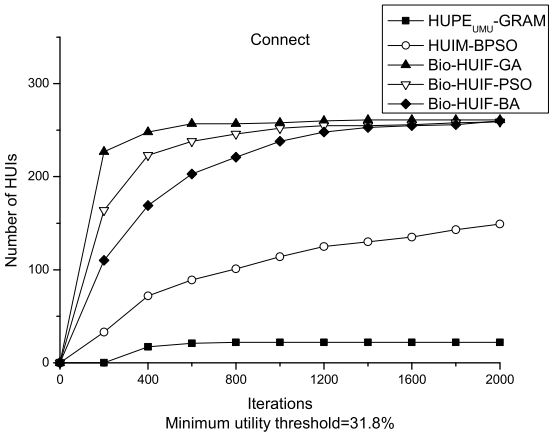


FIGURE 11. Convergence performance comparison for the Connect dataset.

The results of this set of experiments are consistent with the experimental results on the number of discovered HUIs.

E. SUMMARY OF EXPERIMENTAL RESULTS

We now summarize the experimental results from the perspectives of efficiency, the number of discovered HUIs, and convergence speed.

For the experiments on runtime, we can see that both Bio-HUIF-PSO and Bio-HUIF-BA are more efficient than all the other algorithms, including the two exact HUIM algorithms, IHUP and UP-Growth. This shows that bio-inspired algorithms do not need to scan the database many times or construct trees or other structures to transform the original database, and can solve the HUIM problem within a reasonable time. Furthermore, the bitmap representation, bitwise operation, and PEVC used in Bio-HUIF accelerate the process of traversing the search space.

For the experiments on number of discovered HUIs, we can see the other remarkable feature of the three proposed algorithms: they can discover nearly all HUIs in most scenarios. This is obviously better than the other bio-inspired HUIM algorithms. Bio-HUIF-GA is not as efficient as the other two proposed algorithms, and is slower than the other GA-based HUIM algorithm with the Chess, Accidents_10%, and Connect datasets. Bio-HUIF-GA can discover all HUIs under all the listed thresholds for all four datasets. This is because the optimal values of the current population are not retained in the next population under Bio-HUIF. Instead, roulette wheel selection is applied to all discovered HUIs to determine the optimal values of the next population. Thus, the diversity of the population is greatly improved. In contrast, both HUPE_{UMU}-GARM and HUIM-BPSO follow the standard routine of GA or PSO and, accordingly, can only find limited results.

For the experiments on convergence, we can see that the three proposed algorithms can obviously discover more HUIs than HUPE_{UMU}-GARM and HUIM-BPSO within fewer number of iterations. The results on convergence can verify that the search space of HUIs can be traversed efficiently by randomly changing optimal values iteration by iteration. Thus, the improved diversity of the population do accelerate the convergence speed.

VIII. CONCLUSION

In this paper, we have proposed an HUIM framework, Bio-HUIF, from the perspective of bio-inspired algorithms. To fit the scenario of HUIM, all discovered HUIs have a chance of being the optimal values of the next population according to the roulette wheel selection process. Under Bio-HUIF, the bitmap representation of the database, PEVC checking strategy and population initialization method are applied in the three proposed algorithms based on GA, PSO and BA. Experimental results show that the proposed algorithms not only outperform other state-of-the-art bio-inspired HUIM algorithms in terms of efficiency, the number of discovered HUIs, and convergence speed, but are also more efficient than two exact HUIM algorithms.

In future work, we plan to incorporate other bio-inspired algorithms such as bee swarm optimization into Bio-HUIF. Furthermore, the parallelization of Bio-HUIF and related HUIM algorithms will be examined using MapReduce or Spark.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.

[2] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.

[3] F. Benites and E. Sapozhnikova, "Evaluation of hierarchical interestingness measures for mining pairwise generalized association rules," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 3012–3025, Dec. 2014.

[4] R. Chan, Q. Yang, and Y.-D. Shen, "Mining high utility itemsets," in *Proc. 3rd IEEE Intl. Conf. Data Mining*, Nov. 2003, pp. 19–26.

[5] Y. Djenouri and M. Comuzzi, "Combining Apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem," *Inf. Sci.*, vol. 420, pp. 1–15, Dec. 2017.

[6] P. Fournier-Viger *et al.*, "The SPMF open-source data mining library version 2," in *Proc. 19th Eur. Conf. Principles Data Min. Knowl. Discovery*, 2016, pp. 36–40.

[7] J. Gou, F. Wang, and W. Luo, "Mining fuzzy association rules based on parallel particle swarm optimization algorithm," *Intell. Autom. Soft Comput.*, vol. 21, no. 2, pp. 147–162, Apr. 2015.

[8] S.-M. Guo and H. Gao, "HUITWU: An efficient algorithm for high-utility itemset mining in transaction databases," *J. Comput. Sci. Technol.*, vol. 31, no. 4, pp. 776–786, Jul. 2016.

[9] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, Jan. 2004.

[10] K. E. Heraguemi, N. Kamel, and H. Drias, "Association rule mining based on bat algorithm," *J. Comput. Theor. Nanosci.*, vol. 12, no. 7, pp. 1195–1200, Jul. 2015.

[11] K. E. Heraguemi, N. Kamel, and H. Drias, "Multi-swarm bat algorithm for association rule mining using multiple cooperative strategies," *Appl. Intell.*, vol. 45, no. 4, pp. 1021–1033, Dec. 2016.

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[13] J.-P. Huang, C.-T. Yang, and C.-H. Fu, "A genetic algorithm based searching of maximal frequent itemsets," in *Proc. Int. Conf. Artif. Intell.*, 2004, pp. 548–554.

[14] S. Kannimuthu and K. Premalatha, "Discovery of high utility itemsets using genetic algorithm with ranked mutation," *Appl. Artif. Intel.*, vol. 28, no. 4, pp. 337–359, Apr. 2014.

[15] A. K. Kar, "Bio inspired computing—A review of algorithms and scope of applications," *Expert Syst. Appl.*, vol. 59, pp. 20–32, Oct. 2016.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[17] G.-C. Lan, T.-P. Hong, and V. S. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *Knowl. Inf. Syst.*, vol. 38, no. 1, pp. 85–107, Jan. 2014.

[18] Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 198–217, Jan. 2008.

[19] M.-Y. Lin, T.-F. Tu, and S.-C. Hsueh, "High utility pattern mining using the maximal itemset property and lexicographic tree structures," *Inf. Sci.*, vol. 215, pp. 1–14, Dec. 2012.

[20] 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*, vol. 5, pp. 12927–12940, 2017.

[21] 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 Comput.*, vol. 21, no. 17, pp. 5103–5121, Sep. 2017.

[22] J. C.-W. Lin *et al.*, "Mining high-utility itemsets based on particle swarm optimization," *Eng. Appl. Artif. Intell.*, vol. 55, pp. 320–330, Oct. 2016.

[23] Y. Liu, W.-K. Liao, and A. N. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proc. 9th Pacific–Asia Conf. Adv. Knowl. Discovery Data Mining*, 2005, pp. 689–695.

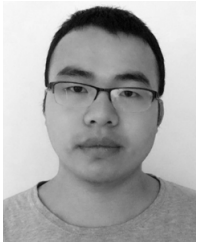
[24] D. Martín, J. Alcalá-Fdez, A. Rosete, and F. Herrera, "NICGAR: A Niching Genetic Algorithm to mine a diverse set of interesting quantitative association rules," *Inf. Sci.*, vols. 355–356, pp. 208–228, Aug. 2016.

[25] R. Pears and Y. S. Koh, "Weighted association rule mining using particle swarm optimization," in *Proc. PAKDD Workshop Bio-Inspired Technol. Data Mining*, 2011, pp. 327–338.

[26] M. Quadrana, A. Bifet, and R. Gavaldà, "An efficient closed frequent itemset miner for the MOA stream mining system," *AI Commun.*, vol. 28, no. 1, pp. 143–158, Jan. 2015.

[27] W. Song, Y. Liu, and J. Li, "BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap," *Int. J. Data Warehousing*, vol. 10, no. 1, pp. 1–15, Jan. 2014.
[28] W. Song, B. Yang, and Z. Xu, "Index-BitTableFI: An improved algorithm for mining frequent itemsets," *Knowl.-Based Syst.*, vol. 21, no. 6, pp. 507–513, Aug. 2008.
[29] T. Tran, B. Vo, T. T. N. Le, and N. T. Nguyen, "Text clustering using frequent weighted utility itemsets," *Cybern. Syst.*, vol. 48, no. 3, pp. 193–209, Mar. 2017.
[30] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1772–1786, Aug. 2013.
[31] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining top-K high utility itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 54–67, Jan. 2016.
[32] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.
[33] X.-S. Yang, "Bat algorithm for multi-objective optimization," *Int. J. Bio-Inspired Comput.*, vol. 3, no. 5, pp. 267–274, Sep. 2011.
[34] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proc. 4th SIAM Int. Conf. Data Mining*, 2004, pp. 482–486.
[35] M. J. Zaki, *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[36] C. Zhang and S. Zhang, *Association Rule Mining: Models and Algorithms*. Berlin, Germany: Springer-Verlag, 2002.

WEI SONG received the Ph.D. degree in computer science from the University of Science and Technology Beijing, Beijing, China, in 2008. He is currently a Professor with the College of Computer Science and Technology, North China University of Technology. His research interests are in the areas of data mining and knowledge discovery. He has authored over 30 research papers in refereed journals and international conferences.



CHAOMIN HUANG received the B.E. degree from Harbin Engineering University, Harbin, China, in 2014. He is currently pursuing the M.E. degree with the College of Computer Science and Technology, North China University of Technology, Beijing, China. His research interests are in the areas of data mining and knowledge discovery.