Soft Comput
DOI 10.1007/s00500-016-2106-1

METHODOLOGIES AND APPLICATION

A binary PSO approach to mine high-utility itemsets

Jerry Chun-Wei Lin[1] · **Lu Yang**[1] · **Philippe Fournier-Viger**[2] · **Tzung-Pei Hong**[3,4] · **Miroslav Voznak**[5]

Abstract High-utility itemset mining (HUIM) is a critical issue in recent years since it can be used to reveal the profitable products by considering both the quantity and profit factors instead of frequent itemset mining (FIM) or association-rule mining (ARM). Several algorithms have been presented to mine high-utility itemsets (HUIs) and most of them have to handle the exponential search space for discovering HUIs when the number of distinct items and the size of database are very large. In the past, a heuristic HUPE$_{umu}$-GRAM algorithm was proposed to mine HUIs based on genetic algorithm (GA). For the evolutionary computation (EC) techniques of particle swarm optimization (PSO), it only requires fewer parameters compared to the GA-based approaches. Since the traditional PSO mechanism is used to handle the continuous problem, in this paper, the discrete PSO is adopted to encode the particles as the binary variables. An efficient PSO-based algorithm, namely HUIM-BPSO, is proposed to efficiently find HUIs. The designed HUIM-BPSO algorithm finds the high-transaction-weighted utilization 1-itemsets (1-HTWUIs) as the size of the particles based on transaction-weighted utility (TWU) model, which can greatly reduce the combinational problem in evolution process. The sigmoid function is adopted in the updating process of the particles for the designed HUIM-BPSO algorithm. An OR/NOR-tree structure is further developed to reduce the invalid combinations for discovering HUIs. Substantial experiments on real-life datasets show that the proposed algorithm outperforms the other heuristic algorithms for mining HUIs in terms of execution time, number of discovered HUIs, and convergence.

Keywords Binary PSO · OR/NOR-tree · Discrete PSO · High-utility itemsets · TWU model

Communicated by V. Loia.

✉ Jerry Chun-Wei Lin
jerrylin@ieee.org

Lu Yang
luyang@ikelab.net

Philippe Fournier-Viger
philfv@hitsz.edu.cn

Tzung-Pei Hong
tphong@nuk.edu.tw

Miroslav Voznak
miroslav.voznak@vsb.cz

1 School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China

2 School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China

3 Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, ROC

4 Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC

5 Department of Telecommunications, Faculty of Electrical Engineering and Computer Science, VSB Technical University of Ostrava, Ostrava-Poruba, Czech Republic

1 Introduction

Knowledge discovery in database (KDD) is an emerging issue since the potential or implicit information can be found from a very large database. Most of them, frequent itemset mining (FIM) or association-rule mining (ARM) has been extensively developed to mine the set of frequent itemsets in which their occurrence frequencies are no less than minimum support threshold or their confidences are no less than minimum confidence threshold (Agrawal and Srikant 1994; Chen et al. 1996). Since only the occurrence frequencies of

Published online: 09 March 2016

itemsets are discovered whether in FIM or ARM, it is insufficient to identify the high-profit itemsets especially when the itemsets are rarely appeared but have high profit values. For example, bread may be purchased hundreds or thousands per day while fewer diamonds are bought in a week or month. The former one has higher frequency but bought with lower profit value while the later one has lower frequency with higher profit value for retailers.

To solve the limitation of FIM or ARM, high-utility itemset mining (HUIM) (Ahmed et al. 2009; Yao et al. 2004; Yao and Hamilton 2006; Yen and Lee 2007) was designed to discover the "useful" and "profitable" itemsets from the quantitative databases. An itemset is considered as a high-utility itemset (HUI) if its utility value is no less than the user-specific minimum utility threshold. In real-world practice, the utility of an itemset can be measured by various factors, such as weight, profit, or cost, which can be defined by users' preferences. Many algorithms were, respectively, developed to mine the set of complete HUIs. Chan et al. (2003) first mentioned utility mining problem instead of FIM. Yao et al. (2004) considered the quantity of items as the internal utility and the unit profit of items as the external utility to discover the HUIs. Since the past algorithms of HUIM suffer the "combinational problem" for discovering HUIs, Liu et al. designed the two-phase (TWU) model and developed the transaction-weighted downward closure (TWDC) property for mining HUIs (Liu et al. 2005). Ahmed et al. adopted the TWU model and designed a IHUP algorithm for interactive and incremental mining HUIs (Ahmed et al. 2009). Lin et al. developed the condensed high-utility pattern (HUP)-tree and related algorithm for discovering HUIs (Lin et al. 2011) by adopting the FP-tree structure (Han et al. 2004) and TWU model. Lan et al. designed the mining algorithm based on index-projection mechanism and developed the pruning strategy to efficiently mine the HUIs (Lan et al. 2013). Tseng et al. then designed the UP-growth mining algorithm (Tseng et al. 2010) to retrieve the HUIs based on the developed UP-tree structure. An efficient list-based algorithm (HUI-Miner) was also proposed to mine the HUIs without candidate generation (Liu and Qu 2012). Other related works of HUIM are still developed in progress (Fournier-Viger et al. 2014; Wu et al. 2012; Zihayat and An 2014).

The traditional algorithms of HUIM have to handle the "exponential problem" of a very huge search space while the number of distinct items or the size of database is very large. Evolutionary computation is an efficient way and able to find the optimal solutions using the principles of natural evolution (Cattral et al. 2009). The genetic algorithm (GA) (Holland 1975) is an optimization approach to solve the NP-hard and non-linear problems and used to investigate a very large search spaces to find the optimal solutions based on the designed fitness functions with various operators such as selection, crossover, and mutation. In the past, Kannimuthu and Premalatha adopted the genetic algorithm and developed the high-utility pattern extracting using genetic algorithm with ranked mutation using minimum utility threshold ($HUPE_{umu}$-GRAM) to mine HUIs (Kannimuthu and Premalatha 2014). Another algorithm, called $HUPE_{wumu}$-GRAM, was also designed to mine HUIs without specific minimum utility threshold (Kannimuthu and Premalatha 2014). For those two algorithms, the crossover and mutation operations are required to randomly generate the next solutions in the evolution process. Besides, it needs amounts of computations to find the satisfied high-utility itemsets in the initial step, which is insufficient when the number of distinct items is very large.

Instead of GAs, particle swarm optimization (PSO) (Kennedy and Eberhart 1995) is also a bio-inspired and population-based approach for finding the optimal solutions by adopting the velocity to update the particles. It uses personal best solutions (*pbest*) and global best solutions (*gbest*) to find the optimal solutions, which is unnecessary to perform the crossover and mutation operations used in GAs. Traditionally, PSO is used to handle the continuous problem to find the optimal solutions, which is insufficient in real-world applications to optimize for a discrete-valued search spaces. Thus, the binary (discrete) PSO algorithm was designed to find the optimal solutions for handling the discrete-valued search spaces (Kennedy and Eberhart 1997). Since each dimension in the particle can be represented as 1 or 0 in BPSO approach, an item can be represented by 1 if it is purchased in the databases or represented as 0 if it is absent in the original databases. Thus, the BPSO-based mechanism (Kennedy and Eberhart 1997) can be used in the issue of HUIM. In this paper, a BPSO-based algorithm with an improved OR/NOR-tree structure is designed for mining the HUIs. The key contributions of this paper are described below:

1. Fewer algorithms have been developed to find the HUIs based on evolutionary computation. In this paper, a discrete PSO-based algorithm, namely HUIM-BPSO, is thus designed to find the HUIs by integrating the sigmoid updating strategy and TWU model.
2. An OR/NOR-tree structure is developed to reduce the multiple database scans by early pruning the invalid combinations of the particles. This process can greatly reduce the computations of the invalid particles (not existing in the original database).
3. Extensive experiments were conducted on real-life datasets to evaluate the performance of the proposed approach. Results showed that the proposed approach can efficiently identify the complete HUIs from very condense databases and outperform the state-of-the-art GA-based algorithms.

The rest of this paper is organized as follows: Related work is briefly reviewed in Sect. 2. Preliminaries and the problem statement of PSO and HUIM are presented in Sect. 3. The proposed HUIM-BPSO algorithm and the designed OR/NOR-tree structure are described in Sect. 4. An illustrated example is presented in Sect. 5. Experiments are conducted and provided in Sect. 6. Finally, conclusions are given in Sect. 7.

2 Related work

In this section, works related to particle swarm optimization (PSO) and high-utility itemset mining (HUIM) are briefly reviewed.

2.1 Particle swarm optimization

In the past, many heuristic algorithms have been facilitated to solve the optimization problems for discovering the necessary information in the evolutionary computation (Cattral et al. 2009; Martnez-Ballesteros et al. 2010). The simple genetic algorithm (SGA) (Holland 1975) is a fundamental search technique to find the feasible and optimal solution in a limit amount of time, which was inspired by the Darwinian principles of the biological evolution, re-producing and the survival of the fittest. Each chromosome in GA is composed of the set of the fixed-length genes as a solution. Three operators such as selection, crossover, and mutation are required in the evolution process of GA. Many variants of GAs have been extensively studied and applied to a wide range of optimization problems (Kannimuthu and Premalatha 2014; Martnez-Ballesteros et al. 2010; Salleb-Aouissi et al. 2007).

Kennedy and Eberhart first introduced particle swarm optimization (PSO) (Kennedy and Eberhart 1995) in 1995, which was inspired by the flocking behavior of birds to solve the optimization problems. Many individuals (particles) are stochastically generated in the search space of PSO in the evolution process. Each particle is represented as an optimized solution by composing a set of velocity in the evolution process. Instead of GA, each particle has memory to keep its previous best particle (personal best, *pbest*) and its previous best particle by considering its neighborhoods (global best, *gbest*). The updating process of the particles based on PSO is decided by the velocity, which is an easier way for implementation and faster to find the optimal solutions compared to the GA-based approach. Besides, it is a non-trivial task to set the appropriate rates of crossover and mutation for GA-based approach. Moreover, PSO is not sensitive to the number of population, which indicates that the PSO-based approach can still find faster the optimal solutions when the number of the population is set lower. The steps of PSO can be illustrated as follows:

1. Initialize the velocities of particles by randomization.
2. The velocity of each particle in the dimension of search space is thus adjusted and updated based on the *gbest* and *pbest*.
3. Each particle in the dimension of search space is thus adjusted and updated by its previous results and the updated velocity.
4. Each updated particle is then evaluated by the pre-defined fitness function to update *pbest* and *gbest* for next population.
5. This iteration is repeatedly processed until the termination criteria is reached.

The PSO was originally defined to solve the continuous valued spaces. In the updating process of PSO, the corresponding particle and velocity are described as follows:

$$v^{id}(t+1) = w_1 \times v^{id}(t) + c_1 \times rand() \times (pbest - x^{id}(t)) + c_2 \times rand() \times (gbest - x^{id}(t)). \quad (1)$$

$$x^{id}(t+1) = x^{id}(t) + v^{id}(t). \quad (2)$$

In Eqs. (1) and (2), t represents current number of iterations; w_1 plays a balancing role between global search and local search; $v^{id}(t)$ is represented the id-th particle velocity; $x^{id}(t)$ is represented the id-th particle; $rand()$ is the random number in range of (0, 1); c_1 is the individual factor; and c_2 is the social factor, which is usually set as 2. In the past, PSO has been adopted to various real-world applications. Kuo et al. designed an algorithm to mine the association rules (ARs) from the investor's stock purchase behavior using the IR value instead of the specific minimum support and minimum confidence thresholds (Kuo et al. 2011). Pears and Koh presented a feasible method to mine the weighted association rules based on PSO (Pears and Koh 2012). More meaningful weights are assigned to the items for revealing useful weighted ARs. For the traditional PSO algorithm, it was designed to handle the continuous-valued search spaces, which could not be used to optimize for a discrete-valued search spaces. In real-world situations, many problems are set as the discrete variable spaces such as scheduling and routing problems. Kennedy and Eberhart then also designed a discrete (binary) PSO (BPSO) (Kennedy and Eberhart 1997) to solve the limitation of continuous PSO. Each particle in BPSO is represented as a set of binary variables. The velocity in the BPSO is updated by the probabilities of sigmoid function. Sarath and Ravi then applied the BPSO optimization approach to discover ARs (Sarath and Ravi 2013). Liang et al. proposed an adaptive PSO based on clustering, by considering the population topology and individual behavior control together to balance local and global search in an optimization process (Liang et al. 2014). Li et al. proposed a global

optimization algorithm inspired by PSO and based on the cuckoo search (CS) algorithm to solve a wide range of real-world optimization problems (Li and Yin 2015). Tsai et al. proposed a high-performance method to solve the clustering problem based on PSO (Tsai et al. 2015). Other applications by adopting PSO to mine the required information are still in progress (Agrawal and Silakari 2013; Menhas et al. 2011; Nouaouria et al. 2013). Besides of PSO, several evolutionary algorithms such as differential evolution (Gong et al. 2010; Storn and Price 1997) were developed and designed for solving the optimization problem.

2.2 High-utility itemset mining

High-utility itemset mining (HUIM) (Yao et al. 2004; Yao and Hamilton 2006) is a critical issue and an emerging topic in recent decades, which can be concerned as the extension of frequent itemset mining (FIM), but more factors such as quantity and profit are considered in it. The purpose of HUIM is to discover the complete set of high-utility itemsets (HUIs) in which the utility of an itemset is no less than the pre-defined minimum utility threshold. The discovered HUIs are concerned as the profitable itemsets which can be used to aid managers or decision makers for making the efficient sales strategies. Chan et al. first developed a mining framework to discover the top-k closed utility patterns (Chan et al. 2003). Not only the complete set of HUIs but also the FIs are discovered based on their designed approach. Yao et al. then designed an approach to discover the profitable itemsets by considering both the purchase quantity (also considered as internal utility) and profit (also considered as external utility) of items to reveal HUIs (Yao et al. 2004; Yao and Hamilton 2006). queryPlease consider rephrasing the following sentence: Since the above algorithms suffer the "combinational problem" to discover HUIs, Liu et al. then developed a two-phase (TWU) model and designed the transaction-weighted downward closure (TWDC) property to early prune the unpromising HUIs but still can discover the complete HUIsSince the above algorithms suffer the "combinational problem" to discover HUIs, Liu et al. then developed a two-phase (TWU) model and designed the transaction-weighted downward closure (TWDC) property to early prune the unpromising HUIs but still can discover the complete HUIs (Liu et al. 2005). To facilitate the problem of interactive and incremental HUIM, Ahmed et al. then developed an IHUP algorithm with a designed tree structure to mine HUIs (Ahmed et al. 2009). In the past, Lin et al. designed a high-utility-pattern (HUP)-tree for discovering HUIs (Lin et al. 2011). It first finds the high-transaction-weighted utilization 1-itemsets (1-HTWUIs) based on TWU model. The kept 1-HTWUIs are then used to build the HUP-tree based on the FP-tree-like approach (Han et al. 2004). Tseng et al. presented the UP-tree structure and UP-growth (Tseng et al. 2010) algorithm for discovering HUIs.

Since the tree-based algorithms limit the memory usage, Lan et al. developed an projection-based algorithm based on the index mechanism to fast mine HUIs (Lan et al. 2013. A novel list-based algorithm, called HUI-Miner, was also developed to mine HUIs without candidate generation, which requires less memory usage but can still efficiently retrieve the HUIs (Liu and Qu 2012). Each entity of a designed utility-list structure keeps three elements: transaction ids (*tids*), the utility value of an item in a transaction (*iutil*), and the resting utility value of items in a transaction (*rutil*). Fournier-Viger et al. then presented an improved algorithm, namely FHM, to quickly mine HUIs based on the built Estimated Utility Co-occurrence Structure (EUCS) (Fournier-Viger et al. 2014). The EUCS keeps the relationships of 2-itemsets, which can be used to reduce the computations of database scans. Other related algorithms are also developed in the progress (Wu et al. 2012; Zihayat and An 2014). Instead of traditional HUIM, Kannimuthua and Premalatha first designed the GA-based algorithm to mine HUIs with the ranked mutation (Kannimuthu and Premalatha 2014). In their approach, it is not easy to find the initial 1-HTWUIs as the chromosome to find HUIs. A very huge computation is necessary to initially set the appropriate chromosomes. Moreover, the parameters of crossover and mutation are required in the evolution process of GAs. In this paper, a high-utility itemset mining based on binary particle swarm optimization (HUIM-BPSO) algorithm is developed and an OR/NOR-tree structure is further designed to avoid the invalid combinations, thus improving the efficiency to discover HUIs.

3 Preliminaries and problem statement

3.1 Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set of m distinct items. A quantitative database is a set of transactions $D = \{T_1, T_2, \ldots, T_n\}$, where each transaction $T_q \in D$ $(1 \leq q \leq n)$ is a subset of I and has a unique identifier q, called its *TID*. Besides, each item i_j in a transaction T_q has a purchase quantity (internal utility) denoted as $q(i_j, T_q)$. Each profit of an item i_j is represented as $pr(i_j)$, and shown in a profit table. A set of k distinct items $X = \{i_1, i_2, \ldots, i_k\}$ such that $X \subseteq I$ is said to be a k-itemset, where k is the length of the itemset. An itemset X is said to be contained in a transaction T_q if $X \subseteq T_q$. A minimum utility threshold is set as δ according to users' preference.

An illustrated example is stated in Table 1 as the running example in this paper. In Table 1, it has ten transactions and six distinct items, denoted from (a) to (f). The profit value

Table 1 A quantitative database

TID	Transaction (item, quantity)
T_1	a:1, c:18, e:1,
T_2	b:6, d:1, e:1, f:1
T_3	a:2, c:1, e:1
T_4	d:1, e:1
T_5	c:4, e:2
T_6	b:1, f:1
T_7	b:10, d:1, e:1
T_8	a:3, c:25, d:3, e:1
T_9	a:1, b:1, f:3
T_{10}	b:6, c:2, e:2, f:4

Table 2 A profit table

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

(external utility) of each item is shown in Table 2 as the profit table. The minimum utility threshold is set as ($\delta = 30\,\%$).

Definition 1 The utility of an item i_j in a transaction T_q is denoted as $u(i_j, T_q)$ and is defined as

$$u(i_j, T_q) = q(i_j, T_q) \times pr(i_j). \tag{3}$$

For example, the utility of items (a), (c) and (e) in transaction T_1 are, respectively, calculated as $u(a, T_1) = q(a, T_1) \times pr(a) = 1 \times 3(=3)$, $u(c, T_1) = q(c, T_1) \times pr(c) = 18 \times 1$ (=18), and $u(e, T_1) = q(e, T_1) \times pr(e) = 1 \times 6(=6)$.

Definition 2 The utility of an itemset X in transaction T_q is denoted as $u(X, T_q)$, and defined as

$$u(X, T_q) = \sum_{i_j \subseteq X \wedge X \subseteq T_q} u(i_j, T_q). \tag{4}$$

For example, the utility of the itemsets (ac) and (ace) in transaction T_1 are, respectively, calculated as $u(ac, T_1) = u(a, T_1) + u(c, T_1) = q(a, T_1) \times pr(a) + q(c, T_1) \times pr(c) = 1 \times 3 + 18 \times 1 (= 21)$ and $u(ace, T_1) = u(a, T_1) + u(c, T_1) + u(e, T_1) = q(a, T_1) \times pr(a) + q(c, T_1) \times pr(c) + q(e, T_1) \times pr(e) = 1 \times 3 + 18 \times 1 + 1 \times 6 (= 27)$.

Definition 3 The utility of an itemset X in a database D is denoted as $u(X)$, and defined as

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q). \tag{5}$$

For example, the utility of itemsets (b) and (bc) in D are, respectively, calculated as $u(b) = u(b, T_2) + u(b, T_6) + u(b, T_7) + u(b, T_9) + u(b, T_{10}) = 54+9+90+9+54(=216)$, and $u(bc) = u(bc, T_{10})(=56)$.

Definition 4 The transaction utility of a transaction T_q is denoted as $tu(T_q)$ and defined as

$$tu(T_q) = \sum_{X \subseteq T_q} u(X, T_q). \tag{6}$$

For example, $tu(T_1) = u(a, T_1) + u(c, T_1) + u(e, T_1) = 3 + 18 + 6$ (=27). The resting transactions from T_2 to T_{10} are, respectively, calculated as $tu(T_2)(=66)$, $tu(T_3)$ (=13), $tu(T_4)(=11)$, $tu(T_5)$ (=16), $tu(T_6)$ (=10), $tu(T_7)$ (=101), $tu(T_8)(=55)$, $tu(T_9)$ (=15), and $tu(T_{10})$ (=72).

Definition 5 The total utility of a database D is denoted as TU and defined as:

$$TU = \sum_{T_q \in D} tu(T_q). \tag{7}$$

For example, the total utility in a database D is calculated as $TU = 27 + 66 + 13 + 11 + 16 + 10 + 101 + 55 + 15 + 72(= 386)$.

Definition 6 An itemset X in a database D is a high-utility itemset (HUI) iff its utility is no less than the minimum utility count as

$$HUI \leftarrow \{X | u(X) \geq TU \times \delta\}. \tag{8}$$

For example, the utility of itemsets (b) and (bc) are, respectively, calculated as $u(b)$ (=216) and $u(bc)$ (=56). Thus, the itemset (b) is a HUI since $u(b) = 216 > 386 \times 0.3 = 115.8$. The itemset ($bc$) is not a HUI since $u(bc) = 56 < 115.8$.

3.2 Problem statement

Based on the above definitions, we define the problem of HUIM as follows (Yao et al. 2004; Yao and Hamilton 2006): Let D be a quantitative transactional database, its profit table *ptable*, and a user-specific minimum utility threshold δ. The problem of HUIM from D is to find the set of high-utility itemsets, in which the utility of an itemset X is no less than ($TU \times \delta$).

4 Proposed BPSO-based framework of HUIM

In the designed binary PSO (BPSO)-based model (Kennedy and Eberhart 1997) for mining HUIs, it consists of pre-processing, particle encoding, fitness evaluation, and the updating processes to mine HUIs. In the first pre-processing process, the high-transaction-weighted utilization 1-itemsets (1-HTWUIs) (Liu et al. 2005) are first discovered based on TWU model (Liu et al. 2005). This process can greatly reduce the invalid itemsets based on the transaction-weighted downward closure (TWDC) property. In the second particle encoding process, the items of 1-HTWUIs are sorted in their alphabetic-ascending order corresponding to the j-th position of a particle. The particle is thus encoded as the set of binary variables corresponding to the sorted order of 1-HTWUIs. In the fitness evaluation, the particles are then evaluated to find the *pbest* and *gbest* particles in the evolution process. For the last updating process, the particles are correspondingly updated by velocities, *pbest*, *gbest*, and the sigmoid function. If the fitness value of the particle is no less than the minimum utility count, it is concerned as a HUI and put into the set of HUIs. This iteration is repeated until the termination criteria are achieved. After that, the set of HUIs is discovered. Details of the four phases are respectively described below.

4.1 Pre-processing phase

In the designed HUIM-BPSO algorithm, the TWU model (Liu et al. 2005) of traditional HUIM is first adopted to discover high-transaction-weighted utilization 1-itemsets (1-HTWUIs). Based on the transaction-weighted downward closure (TWDC) property of HTWUIs, the unpromising items can be efficiently pruned. Thus, the computations for discovering HUIs can be greatly improved. The phase to discover 1-HTWUIs is described below. First, the utility of items of each transaction is calculated as the transaction utility (*tu*). The transaction-weighted utility of an item is thus calculated by summing up the transaction utility if an item appearing in the transaction. This process is used to estimate the upper-bound value of an item. If the transaction-weighted utility of an item is no less than the minimum utility count, it is considered as a HTWUI. In this example, the minimum utility count is calculated as ($386 \times 0.3 = 115.8$). The discovered 1-HTWUIs are shown in Table 3.

4.2 Particle encoding phase

Each particle in the designed approach is to present a set of itemsets, forming the potential HUI. The size of the particle is the number of 1-HTWUIs found in the first pre-processing phase. Each particle is composed by a set of binary variables as 0 or 1, which indicates that a correspondingly item is present or absent in a particle. If the corresponding j-th position of a particle is set as 1, it indicates that the an item in j-th position is presented as a potential HUI; otherwise, this item is not included and cannot be a potential HUI. Note that the discovered 1-HTWUIs are sorted in alphabetic-ascending order corresponding to the positions in the particle. Initially, the discovered TWU values of 1-HTWUIs will be normalized as the probabilities to initialize the particles in the population. The velocities of particles in a population are randomly generated in the range of (0, 1). In this example, the size of a particle is set as 5, which was shown in Table 3. The particles can be represented in Fig. 1.

Table 3 Discovered 1-HTWUIs

Item	TWU	1-HTWUIs
(*a*)	110	No
(*b*)	264	Yes
(*c*)	183	Yes
(*d*)	233	Yes
(*e*)	361	Yes
(*f*)	163	Yes

	b	*c*	*d*	*e*	*f*
p_1	1	0	0	1	1
p_2	0	1	0	0	1
p_3	0	0	1	0	0
⋮					
p_n	0	1	0	1	0

Fig. 1 Encoding of particles

4.3 Evaluation phase

The fitness function in the designed algorithm is utilized to evaluate the utility value of each particle, which is the same as the traditional HUIM to discover HUIs as

$$\text{Fitness}(p_i) = u(X), \tag{9}$$

in which X is the union of the j-th item in the particle if its value is set as 1. In this example, the itemset of particle p_1 is (*bef*), and the particle p_2 is (*cf*). Thus, if the utility value of a particle is no less than the minimum utility count, it is considered as a HUI and will be put in the set of HUIs.

4.4 Updating phase

After the evaluation process of the particles in a population, the velocities of the particles are updated according to the traditional PSO approach, shown in the Eq. (1). The particles are updated based on *sigmoid* function used in BPSO

approach (Kennedy and Eberhart 1997). The obtained equation for updating the particles is shown as follows:

$$x^{id}(t+1) = \begin{cases} 1 & rand() < sig(v^{id}(t+1)) = \frac{1}{1+e^{-v^{id}(t)}} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

From the above Eq. (10), the *sigmoid* function is used for normalization. The *rand*() function is a uniform distribution in the range of (0, 1). This equation is used to determine the probability of the (j-th) position of a particle. When the generated $rand()$ is less than the $sig(v^{id}(t+1))$, the value of the corresponding j-th position of a particle is set as 1; otherwise, it is set as 0.

4.5 An improved strategy of combination

Although the adopted TWU model (Liu et al. 2005) for generating 1-HTWUIs can be used to eliminate the unpromising HUIs, several redundant and meaningless combinations not existing in the databases are still produced in the evolution process. To improve the efficiency and reduce the invalid combinations of the 1-HTWUIs in the particles, an enumeration OR/NOR-tree is designed and built in the initial step for generating the valid combinations of particles. First, each transaction is then revised to keep only the discovered 1-HTWUIs in *alphabetic-ascending* order. The maximal patterns (itemsets) of the database are then retrieved. This approach can be used to compress the patterns if it is contained within its superset, thus reducing the size of the designed OR/NOR-tree structure.

Definition 7 For two patterns (a) and (b), (a) is considered as a maximal pattern if $(b) \subseteq (a)$.

For the itemset (ce) in T_1 shown in Table 1, (ce) is contained within the itemset (cde) and $(bcef)$; (cde) and $(bcef)$ are the maximal patterns since $(ce) \subseteq (cde)$, and $(ce) \subseteq (bcef)$. In this example, the itemsets $(bdef)$, (cde), and $(bcef)$ are the maximal patterns in D.

After that, the revised transactions are then processed tuple-by-tuple from the first transaction to the last one to construct the OR/NOR-tree structure. The corresponding position of 1-HTWUIs in a particle is sorted as the *alphabetic-ascending* order. This OR/NOR-tree structure is used to determine whether the combined items of a particle exist in the database, which can be used to avoid the redundant combinations in the evolution process. From the given example, the constructed OR/NOR-tree structure is shown in Fig. 2.

In the designed OR/NOR-tree structure, the OR operator indicates that an item can be presented or absent in the particle, in which either value 0 or 1 in the j-th position of a particle will return a *true* value by $\vee$ (union) operator. The NOR operator indicates that an item can be only absent in the particle, in which only 0 value on the j-th position of a particle will return *true* value by $\downarrow$ (NOR) operator. For example, an itemset (bcd) cannot be generated since this particle can be encoded as (11100); it does not match any branches in the developed OR/NOR-tree structure since the second position of a particle should be certainly set as 0 for the maximal pattern as $(bdef)$; the third position of a particle should be certainly set as 0 for the maximal pattern as $(bcef)$, and the first position of a particle should be certainty set as 0 for the maximal pattern as (cde). Based on the designed OR/NOR-tree, the computations of particles in the evolution process can be greatly improved since the invalid combinations can be avoided in the evolution process.

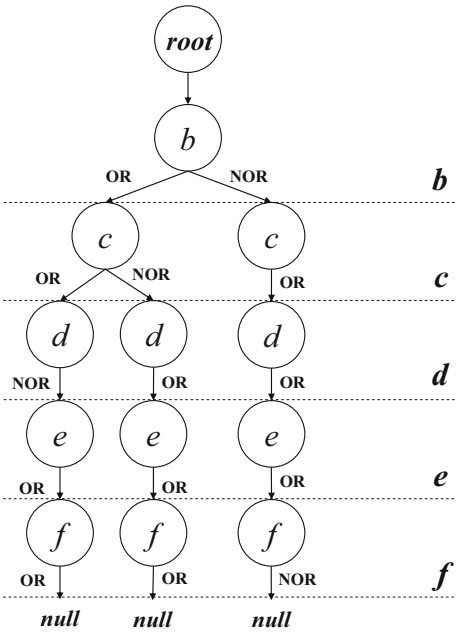

Fig. 2 The designed OR/NOR-tree structure

4.6 The designed algorithm

The above phase is then repeatedly processed until the termination condition is reached. The algorithm of the designed HUIM-BPSO is shown in Algorithm 1.

In the designed HUIM-BPSO algorithm, the set of 1-HTWUIs is first discovered as the size of particles in the evolution process (Lines 1 to 7). The OR/NOR-tree is then constructed according to the discovered maximal patterns (Line 8). The particles are initialized based on the OR/NOR-tree and random number (Line 11). After that, the velocities are also generated by uniform distribution in the range of (0, 1) (Line 12). If the fitness (utility) of the initial particle is no less than the minimum utility count, the items of the particle

Algorithm 1: HUIM-BPSO

Input: D, a quantitative database; *ptable*, a profit table, δ, the minimum utility threshold; M, the number of particles of each iteration.
Output: *HUIs*, a set of high-utility itemsets.

```
1  for each T_q ∈ D do
2      for each i_j ⊆ T_q do
3          tu(T_q) = q(i_j, T_q) × ptable(i_j);
4  calculate twu(i_j) = Σ_{i_j ⊆ T_q} tu(T_q);
5  TU = Σ_{T_q ∈ D} tu(T_q);
6  find 1-HTWUIs ← {i_j | twu(i_j) ≥ TU × δ};
7  set k = |1-HTWUIs| //*
8  set particle size construct OR/NOR-tree;
9  for i ← 1 to M do
10     for j ← 1 to k do
11         initialize p_i^j(t) = either 0 or 1 //*
12         initial the particles according to OR/NOR-tree initialize
             v_i^j(t) = rand() //*
13         initial the velocity of particles in (0, 1)
14     if fitness(p_i(t)) ≥ TU × δ then
15         HUIs ← GetItem(p_i(t)) ∪ HUIs //*
16         find a HUI
17     find pbest(t) of each M particle //*
18     update pbest find gbest(t) among M pbest particles and
         gbest(t) //*
19     update gbest
20 while termination criteria is not reached do
21     for i ← 1, M do
22         update the v_i(t + 1) velocities of M particles //*
23         by equation (1) for j ← 1 to k do
24             check the OR/NOR-tree to return the true value of the
               j-th position of p_i^j(t + 1);
25             update the p_i^j(t + 1) of M particles //*
26             by equation (10)
27         if fitness(p_i(t + 1)) ≥ TU × δ then
28             HUIs ← GetItem(p_i(t + 1)) ∪ HUIs //*
29             find a HUI
30         find pbest(t + 1) of each M particle //*
31         update pbest find gbest(t + 1) among M pbest particles
             and gbest(t) //*
32         update gbest
33     set t ← t + 1;
34 return HUIs;
```

is then outputted and concerned as the high-utility itemset (Lines 13 to 14). The *pbest* and *gbest* are also initialized from the generated particles (Lines 15 to 16). The updating process of velocities is the same as the traditional PSO approach (Line 19). The particles are updated according to the BPSO approach (Kennedy and Eberhart 1997) based on *sigmoid* function and the developed OR/NOR-tree structure to find the valid particles (Lines 21 to 22). If the fitness (utility) of a particle is no less than the minimum utility count, the items of the particle are then outputted and considered as the high-utility itemset (Lines 23 to 24). This process is then repeated until the termination criteria is achieved (Lines 17 to 27). After that, the set of discovered HUIs is then returned (Line 28). The flowchart of the designed algorithm is shown in Fig. 3.

5 An illustrated example

In this section, a database shown in Table 1 and the profit table shown in Table 2 are used as the running example to illustrate the procedure of the designed HUI-BPSO algorithm. The minimum utility threshold is set as $\delta(=30\,\%)$. Thus, the minimum utility count is calculated as $(TU \times \delta)$ $(=386 \times 0.3)$ $(=115.8)$. The quantitative database in Table 1 is first scanned to find the 1-HTWUIs, which was shown in Table 3. Thus, the size of each particle in the designed algorithm is set as 5, which is equal to the number of discovered 1-HTWUIs. Based on the transaction-weighted downward closure (TWDC) property, this process can be used to eliminate the combinational process for discovering HUIs. After that, the number of particles in the population is initially set as 10, as well as the number of iterations. Note that those two parameters can be adjusted by users' preferences.

The maximal patterns in D are then discovered to construct the developed OR/NOR-tree structure. From the running example, the results are $(bdef)$, (cde), and $(bcef)$. The constructed OR/NOR-tree structure was shown in Fig. 2. The initial particles are then randomly generated based on the valid combinations of the OR/NOR-tree structure. The velocity of each particle is also generated in the range of (0, 1). The results of the generated particles and velocities are, respectively, shown in Figs. 4 and 5.

After that, the fitness of each particle in Fig. 4 is then calculated. The results are shown in Table 4.

From the results of Fig. 4, it can be observed that the p_9 has the highest fitness value and larger than the minimum utility count; p_9 is thus set as the global best (*gbest*) in this iteration. The velocity of each particle is thus updated according to the Eq. (1). In this example, the particles p_3, p_6, and p_9 are put in the set of HUIs since their utility values are no less than the minimum utility count. In the given example, c_1 and c_2 are set the same as 1, which can be adjusted by users' preferences. The updated velocity is shown thus in Fig. 6. In this step, the discovered set of HUIs $= \{bef, b\}$.

The particles are then updated based on the designed OR/NOR-tree structure and Eq. (10). For example, the first level of (b) has two branches, which indicates that the item (b) can be either 0 or 1. Thus, a determination of Eq. (10) is applied to set the binary value of (b); if (b) is set as 1, it uses the OR operator (the left side branch of OR/NOR-tree structure shown in Fig. 2) to return the *true* value of (b). Otherwise, (b) is set as 0 and the NOR operator (the right side branch of OR/NOR-tree structure shown in Fig. 2) is

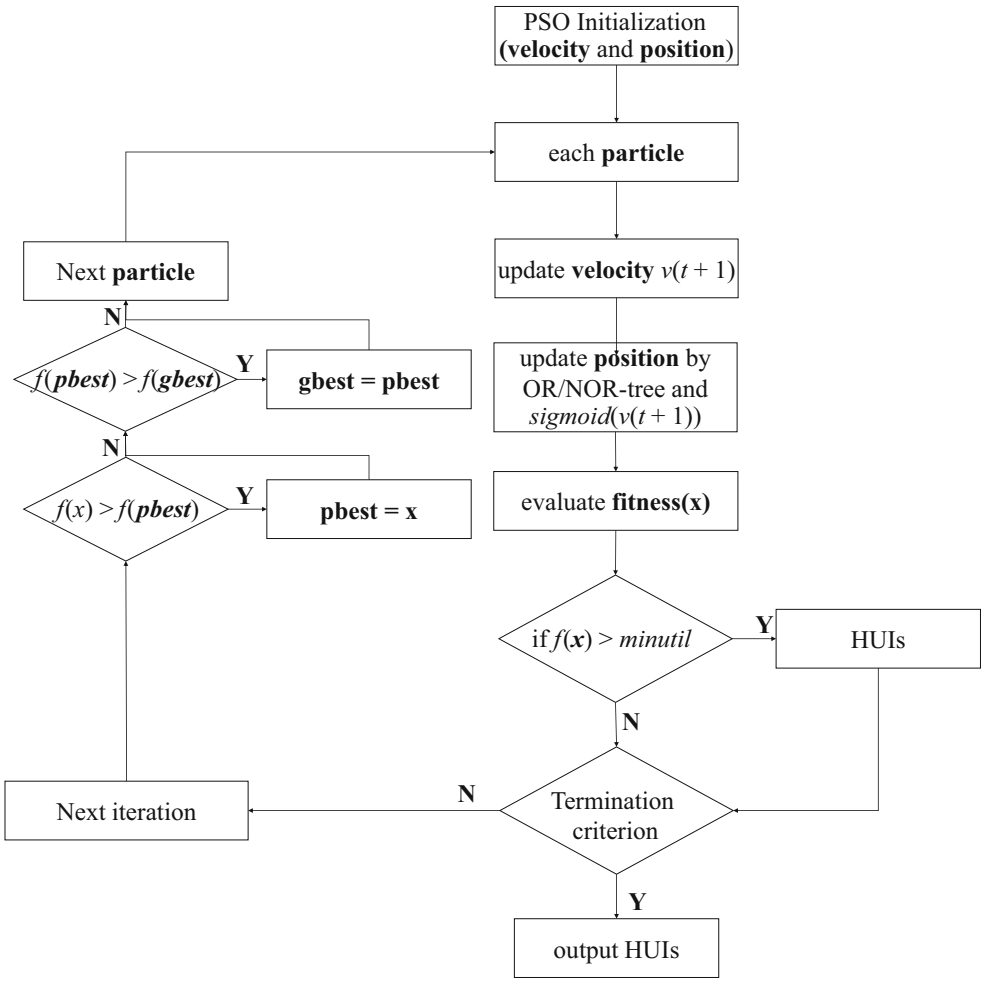

Fig. 3 The flowchart of the designed algorithm

	b	*c*	*d*	*e*	*f*
p_1	0	0	1	0	1
p_2	0	1	0	1	0
p_3	1	0	0	1	1
p_4	0	0	1	1	1
p_5	0	1	0	1	0

	b	*c*	*d*	*e*	*f*
p_6	1	0	0	0	0
p_7	0	0	0	1	1
p_8	0	1	0	1	0
p_9	1	0	0	0	0
p_{10}	0	0	1	1	0

Fig. 4 Initial particles

used to return the *true* value of (*b*). Based on the designed OR/NOR-tree structure, each *j*-th position of a particle must return the *true* (1) value as a valid particle based on the given OR and NOR operators. The results of the updated particles are shown in Fig. 7 and the fitness values of the updated particles are shown in Table 5.

From Table 5, if the fitness value of a particle is no less than minimum utility count, it is concerned as a HUI and put into the set of HUIs. In this example, the fitness values of $p_1(=b)$ and $p_8(=b)$ are no less than minimum utility count (=115.8), but they are the same and already existing in the set of HUIs; nothing has to be done in this step. After that, the evolution and updating process are repeated and corresponding updated until the termination criteria ia achieved. Finally, the discovered HUIs are shown in Table 6.

	b	c	d	e	f
v_1	0.33	0.92	0.26	0.36	0.84
v_2	0.40	0.22	0.74	0.61	0.16
v_3	0.64	0.94	0.76	0.94	0.76
v_4	0.76	0.24	0.53	0.18	0.87
v_5	0.40	0.58	0.83	0.60	0.60

	b	c	d	e	f
v_6	0.74	0.13	0.01	0.61	0.26
v_7	0.75	0.02	0.99	0.84	0.20
v_8	0.67	0.33	0.05	0.51	0.61
v_9	0.01	0.91	0.77	0.99	0.59
v_{10}	0.95	0.08	0.20	0.27	0.57

Fig. 5 Initial velocities

Table 4 Fitness value of each particle

Particle	Item	Fitness
p_1 (=00101)	(*df*)	6
p_2 (=01010)	(*ce*)	92
p_3 (=10011)	(*bef*)	131
p_4 (=00111)	(*def*)	12
p_5 (=01010)	(*ce*)	92
p_6 (=10000)	(*b*)	216
p_7 (=00011)	(*ef*)	23
p_8 (=01010)	(*ce*)	92
p_9 (=10000)	(*b*)	216
p_{10} (=00110)	(*de*)	54

6 Experimental results

Substantial experiments were conducted to verify the effectiveness and efficiency of the proposed the HUIM-BPSO algorithm with and without the developed OR/NOR-tree structure compared to the state-of-the-art HUPE$_{umu}$-GRAM algorithm (Kannimuthu and Premalatha 2014). The HUPE$_{umu}$-GRAM algorithm prunes the unpromising itemsets based on the transaction-weighted downward closure (TWDC) property of HTWUIs in the first step, which is the same with the proposed algorithm. The effective OR/NOR-tree structure is developed of the designed HUIM-BPSO algorithm, which can be used to reduce the invalid combinations of the 1-HTWUIs in the particles. This tree structure can also be adopted in the GA-based HUPE$_{umu}$-GRAM algorithm but only in the pre-processing phase since the updating process of the GA-based algorithm still adopts the mutation and crossover operations to generate the candidates for next iteration. The updating process of the particles for the designed HUIM-BPSO algorithm can be modified bit by bit based on the OR/NOR-tree structure, which is totally different than that of the GA-based approach. Besides, the original HUPE$_{umu}$-GRAM algorithm requires the HUIs as the initial particles for later evolution process, which needs very large computations to find the initial particles. We have thus improved this approach by adopting our designed pre-processing phase to reduce the computations of the traditional HUPE$_{umu}$-GRAM algorithm.

In the conducted experiments, the HUPE$_{umu}$-GRAM algorithm with the designed OR/NOR-tree structure is named as HUPE$_{umu}$ -GRAM^{+}; the original HUPE$_{umu}$-GRAM algorithm without the designed OR/NOR-tree structure is named as HUPE$_{umu}$-GRAM^{-}; the HUIM-BPSO algorithm with the OR/NOR-tree structure is named as HUIM-BPSO^{+}; and the HUIM-BPSO algorithm without OR/NOR-tree structure is named as HUIM-BPSO^{-} algorithm. The algorithms in the experiments were implemented in C++ language, performing on a PC with an Intel Core2 i3-4160 CPU and 4GB of RAM, running the 64-bit Microsoft Windows 7 operating system. Six real-world datasets, called chess (Frequent itemset mining dataset repository 2012), mushroom (Frequent itemset mining dataset repository 2012), connect (Frequent itemset mining dataset repository 2012), accidents (Frequent itemset mining dataset repository 2012), foodmart (Microsoft 1996), and retail (Frequent itemset mining dataset repository 2012), are used in the experiments, which were widely used in the issue of high-utility itemset mining (HUIM) (Fournier-Viger et al. 2014; Fournier-Viger and Zida 2015; Lin et al. 2015, 2011; Liu and Qu 2012; Tseng et al. 2010; Zida et al. 2015). A simulation model (Liu et al. 2005) was developed to generate the quantities and profit values of items in transactions for all datasets except foodmart which already has real utility values. A log-normal distribution was used to randomly assign quantities in the [1,5] interval, and item profit values in the [1,1000] interval. Parameters and characteristics of the datasets used in the experiments are, respectively, shown in Tables 7 and 8.

In the conducted experiments for mining HUIs, the number of populations and iterations is set at 10,000 and the population size is set as 20. We have also ran five times the experiments to show the optimal results from it. Both HUPE$_{umu}$-GRAM^{-} and HUPE$_{umu}$-GRAM^{+} algorithms also adopt the binary encoding mechanism, as well as the designed binary PSO-based algorithm. For the HUIM-BPSO^{+} and HUIM-BPSO^{-} algorithms, w_1 is set as 0.9; the individual

	b	*c*	*d*	*e*	*f*
v_1	0.86	0.92	-0.27	0.36	0.30
v_2	0.54	0.08	0.74	0.47	0.16
v_3	0.64	0.94	0.76	0.50	0.32
v_4	1.56	0.24	-0.26	-0.62	0.06
v_5	0.82	0.16	0.83	0.17	0.60

	b	*c*	*d*	*e*	*f*
v_6	0.74	0.13	0.01	0.61	0.26
v_7	0.87	0.02	0.99	0.72	0.08
v_8	1.40	-0.39	0.05	-0.21	0.61
v_9	0.01	0.91	0.77	0.99	0.59
v_{10}	1.62	0.08	-0.46	-0.39	0.57

Fig. 6 Updated velocities

	b	*c*	*d*	*e*	*f*
p_1	1	0	0	0	0
p_2	0	0	1	0	0
p_3	1	1	0	1	0
p_4	0	1	0	1	0
p_5	0	0	0	1	0

	b	*c*	*d*	*e*	*f*
p_6	0	0	0	1	0
p_7	1	0	1	0	1
p_8	1	0	0	0	0
p_9	1	0	1	1	1
p_{10}	0	1	1	0	0

Fig. 7 Updated particles

Table 5 Fitness value of each updated particle

Particle	Item	Fitness
p_1 (=10000)	(*b*)	216
p_2 (=00100)	(*d*)	30
p_3 (=11010)	(*bce*)	68
p_4 (=01010)	(*ce*)	92
p_5 (=00010)	(*e*)	60
p_6 (=00010)	(*e*)	60
p_7 (=10101)	(*bd*)	60
p_8 (=10000)	(*b*)	216
p_9 (=10111)	(*bdef*)	66
p_{10} (=01100)	(*cd*)	40

Table 6 Discovered HUIs

Particle	Item	Fitness
p_1 (=10010)	(*be*)	222
p_2 (=10000)	(*b*)	216
p_3 (=10110)	(*bde*)	166
p_4 (=10100)	(*bd*)	154
p_5 (=10001)	(*bf*)	135
p_6 (=10011)	(*bef*)	131

factor c_1 and the social factor c_2 are both set as 2. The parameters used in the HUPE$_{\text{umu}}$-GRAM$^+$ and HUPE$_{\text{umu}}$-GRAM$^-$ algorithms, respectively, are roulette wheel selection, one-point crossover, and ranked mutation. The crossover rate is initially set as 0.9 in the experiment. Since the HUPE$_{\text{umu}}$-GRAM$^+$ adopts the ranked mutation mechanism, the mutation rate is dynamically changed based on the ranking score of the chromosomes. The algorithms are then compared in terms of execution time, number of HUIs, number of determined nodes, and the convergence as follows:

Table 7 Parameters of used datasets

#$\|D\|$	Total number of transactions
#$\|I\|$	Number of distinct items
AvgLen	Average transaction length
MaxLen	Maximal length transactions
Type	Dataset type

Table 8 Characteristics of used datasets

Dataset	#$\|D\|$	#$\|I\|$	AvgLen	MaxLen	Type
Chess	3196	76	37	37	Dense
Mushroom	8124	120	23	23	Dense
Connect	67,557	129	43	43	Dense
Accidents_10 %	34,018	469	34	46	Dense
Foodmart	21,557	1559	4	11	Sparse
Retail	88,162	16,470	10	76	Sparse

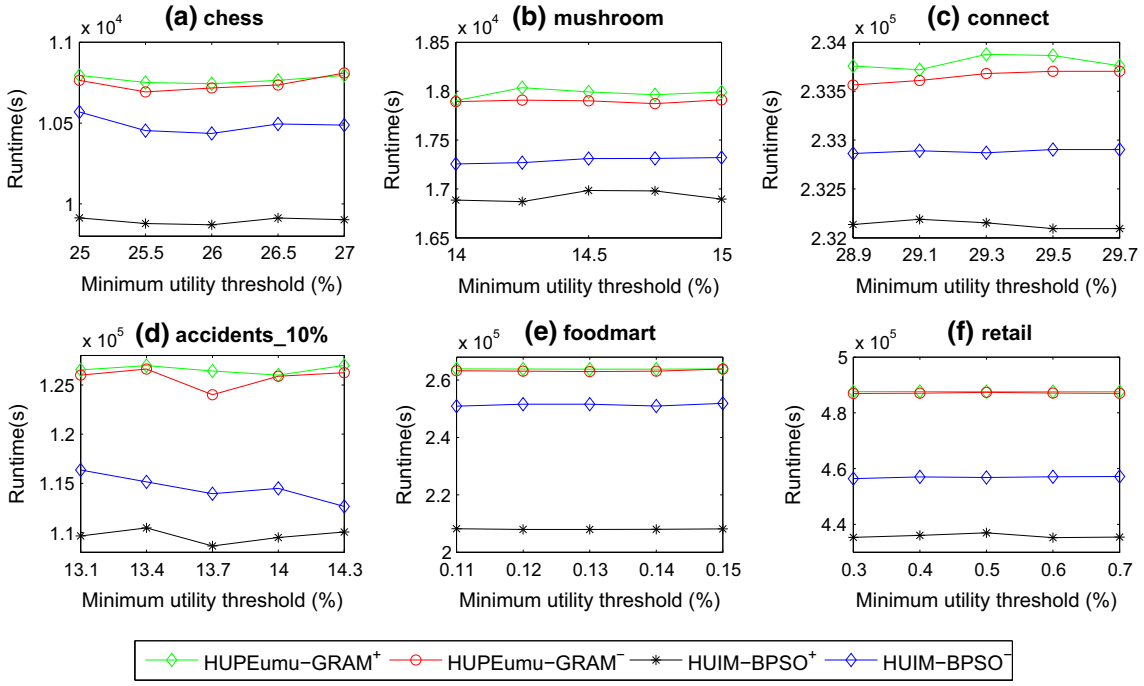

Fig. 8 Runtime w.r.t variants of minimum utility thresholds

6.1 Runtime

In the conducted experiments of runtime in six datasets, the four algorithms are then compared w.r.t. variants of minimum utility thresholds. The results are shown in Fig. 8.

From Fig. 8, it can be seen that both the HUIM-BPSO$^-$ and HUIM-BPSO$^+$ algorithms outperform the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms w.r.t. different minimum thresholds in six datasets. For example in Fig. 8a, the runtimes of the HUPE$_{umu}$-GRAM$^+$, the HUPE$_{umu}$-GRAM$^-$, the HUIM-BPSO$^-$, and the HUIM-BPSO$^+$ were 10,795, 10,764, 10,568, and 9913 seconds at 10,000 iteration in chess dataset. The HUIM-BPSO$^+$ algorithm always has better results than that of the HUIM-BPSO$^-$ algorithm since the HUIM-BPSO$^+$ algorithm only generates the valid combinations of itemsets existing in the database, which can greatly avoid the combinational problem in the evolution process. The HUIM-BPSO$^-$ algorithm still requires, however, to generate the invalid itemsets not existing in the database for discovering HUIs. Overall, the proposed HUIM-BPSO$^+$ has better performance than the improved HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms in six different datasets. From the above results, it also can be found that the HUPE$_{umu}$-GRAM$^+$ algorithm needs more time to construct the OR/NOR-tree structure for later mining process than that of the HUPE$_{umu}$-GRAM$^-$ algorithm.

6.2 Number of HUIs

In this section, the number of HUIs is evaluated to show the performance of the compared algorithms. The TWU model (Liu et al. 2005) is used to discover the actual and complete HUIs from the quantitative databases. Experiments are conducted and shown in Fig. 9.

From Fig. 9a–d, it can be seen that the proposed HUIM-BPSO$^+$ algorithm generates nearly the same number of HUIs as the TWU model especially when the minimum utility threshold is set high in the condense datasets. The reason is that the size of a particle is associated with the number of 1-HTWUIs; less computations are required when the minimum utility threshold is set higher. For more condensed datasets such as chess, mushroom, connect, and accidents datasets, the number of 1-HTWUIs is close to the number of discovered HUIs with higher minimum utility threshold; the number of generated HUIs of the designed HUIM-BPSO$^+$ algorithms is close to the traditional way for mining the complete HUIs. For example, in Fig. 9a under the 25 % minimum utility threshold, the number of HUIs of the HUPE$_{umu}$-GRAM$^+$ algorithm, the HUPE$_{umu}$-GRAM$^-$ algorithm, the HUIM-BPSO$^-$ algorithm, the HUIM-BPSO$^+$ algorithm, and the TWU model were, respectively, found as 11, 12, 75, 83, and 98. The HUIM-BPSO$^+$ and HUIM-BPSO$^-$algorithms have nearly the same number of HUIs as the TWU model when the minimum utility threshold is set higher than 25 %, which can be observed in Fig. 9a. From Fig. 9b–d, it can be

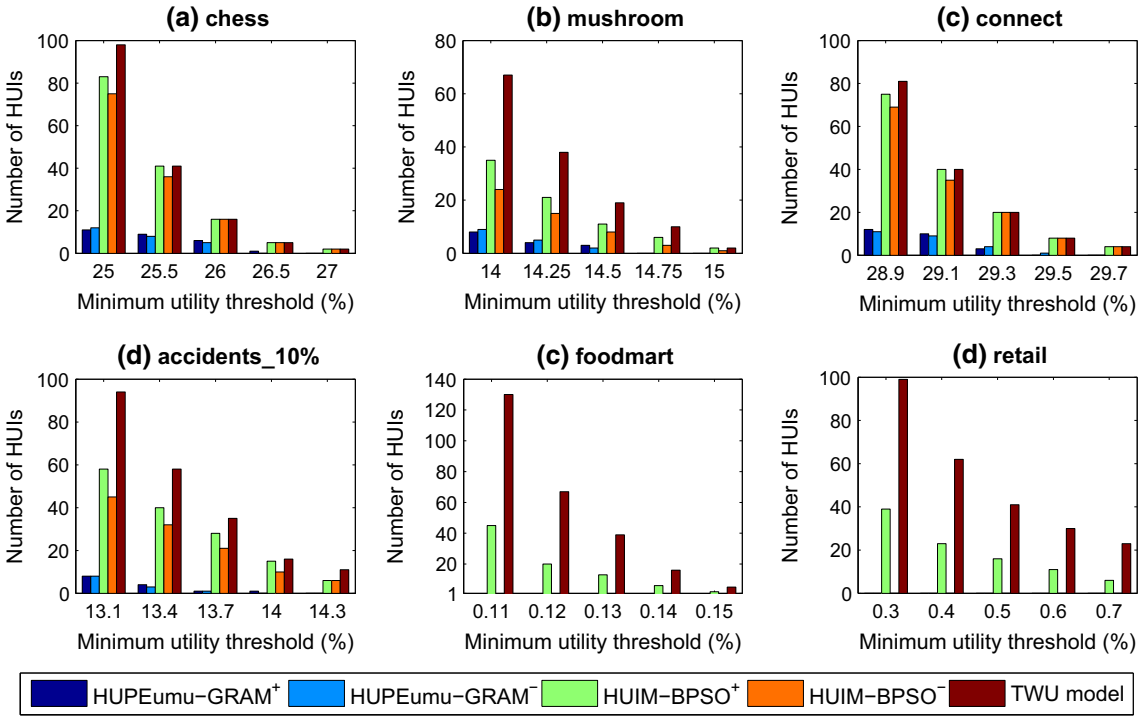

Fig. 9 Number of HUIs w.r.t variants of minimum utility thresholds

observed that the HUIM-BPSO$^+$ algorithm discovers more number of HUIs than the HUPE$_{umu}$-GRAM$^+$ algorithm and the HUPE$_{umu}$-GRAM$^-$ algorithm. From the above experiments, it can be found that results of the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms are similar. The reason is that the OR/NOR-tree structure is useless in the HUPE$_{umu}$-GRAM$^+$ algorithm since it can only be used once in the pre-processing step. The updating process of HUPE$_{umu}$-GRAM$^+$ algorithm still, however, requires the mutation and crossover operations. Thus, the designed OR/NOR-tree structure is useless to the GA-based approach.

For the conducted experiments shown in Fig. 9e, f, it can be obviously found that the number of HUIs for the HUPE$_{umu}$-GRAM$^+$ algorithm, HUPE$_{umu}$-GRAM$^-$ algorithm, and the HUIM-BPSO$^-$ algorithm is close to zero. The HUIM-BPSO$^+$ algorithm can still, however, discover the HUIs in the evolution process. The reason is that for a very sparse datasets such as foodmart dataset, the number of generated 1-HTWUIs is very large but the number of actual HUIs is quite low. The HUIM-BPSO$^-$ algorithm has to generate the potential itemsets of each iteration, thus requiring more computations to discover the actual HUIs. Thus, it can be concluded that the evolutionary algorithms are insufficient to discover the HUIs in the sparse dataset, especially when the minimum utility threshold is set low. The HUIM-BPSO$^+$ algorithm can still, however, discover partial HUIs compared to the TWU model algorithm and still outperform the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms.

From Tables 9 and 10, it showed the accuracy of the number of HUIs discovered by different algorithms comparing with the traditional TWU model. From the results shown in Tables 9 and 10, it is obvious that the designed HUIM-BPSO$^+$ algorithm can mine nearly the same number of the HUIs as the TWU model in the chess and connect datasets, especially when the minimum utility threshold is set higher. For the mushroom and accidents datasets, the HUIM-BPSO$^+$ algorithm can generate more than 50 % of HUIs, but the HUPE$_{umu}$-GRAM$^-$ and HUPE$_{umu}$-GRAM$^+$ algorithms could not generate any HUIs even the minimum utility threshold is set higher. The reason is that the valid HUIs are retrieved from the 1-HTWUIs, and it has higher possibility to generate more HUIs when the minimum utility threshold is set lower of the HUPE$_{umu}$-GRAM$^-$ algorithm. From Figs. 9 and 10, it can be observed that the state-of-the-art HUPE$_{umu}$-GRAM$^-$ algorithm cannot produce the valid HUIs from the 1-HTWUIs under variants of minimum utility threshold. For the foodmart and retail datasets, the proposed HUIM-BPSO$^+$ algorithm can generate about 30 % of HUIs. The HUIM-BPSO$^-$, the HUPE$_{umu}$-GRAM$^+$, and the HUPE$_{umu}$-GRAM$^-$ algorithms, however, could not generate any HUIs. Overall, when the number of iterations is increased, the proposed HUIM-BPSO$^+$ algorithm can achieve higher accuracy to generate the same number of HUIs as the traditional TWU model.

Table 11 shows the average standard scores of experiments running five times w.r.t. variants minimum utility thresholds in six datasets. For example, when the minimum

Table 9 Percentages of discovered HUIs of the BPSO-based approaches

Algorithm		HUIM-BPSO$^+$			HUIM-BPSO$^-$		
Dataset	δ (%)	Best (%)	Average (%)	Worst (%)	Best (%)	Average (%)	Worst (%)
Chess	25	88.77	86.78	84.69	76.53	74.53	69.38
	25.5	100	100	100	95.12	92.24	87.80
	26	100	100	100	100	100	100
	26.5	100	100	100	100	100	100
	27	100	100	100	100	100	100
Mushroom	14	62.68	54.22	49.25	40.29	38.90	35.82
	14.25	68.42	64.15	55.26	47.36	41.47	34.21
	14.5	78.94	65.55	52.63	42.10	35.57	26.31
	14.75	100	83	60	50	42	30
	15	100	100	100	100	100	50
Connect	28.9	98.76	94.82	90.12	83.18	81.48	80.24
	29.1	100	98.50	95	92.50	89.50	85
	29.3	100	100	100	100	100	100
	29.5	100	100	100	100	100	100
	29.7	100	100	100	100	100	100
Accidents	13.1	69.14	67.95	61.70	48.93	47.96	46.80
	13.4	74.13	71.68	68.96	55.17	54.14	50
	13.7	94.28	86.72	80	60	58.14	51.43
	14	100	95.75	81.25	75	69.75	62.50
	14.3	100	82.72	54.54	54.54	47.45	36.36
Foodmart	0.11	38.46	37.15	34.61	0	0	0
	0.12	31.34	30.15	28.35	0	0	0
	0.13	35.89	32.76	28.20	0	0	0
	0.14	43.75	39.50	31.25	0	0	0
	0.15	40	25	0	0	0	0
Retail	0.3	41.41	40.38	36.36	0	0	0
	0.4	37.09	35.87	32.25	0	0	0
	0.5	51.21	48.90	39.02	0	0	0
	0.6	43.33	42	36.66	0	0	0
	0.7	26.08	23.73	13.04	0	0	0

utility threshold is set as 25 %, the average standard score of the discovered HUIs by the HUIM-BPSO$^+$ algorithm is 1.0405, which indicates that the number of HUIs mined by HUIM-BPSO$^+$ has 1.0405 standard deviations of the discovered HUIs among the four algorithms. From Table 11, it can be seen that the all standard scores of the HUIM-BPSO$^+$ algorithm are more than zero, which shows that the number of HUIs mined by the HUIM-BPSO$^+$ is more than the other algorithms. The standard scores of the HUPE$_{umu}$-GRAM$^-$ and the HUPE$_{umu}$-GRAM$^+$ algorithms are all less than zero, which showed that the number of HUIs mined by the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms are all less than the other algorithms. Overall, the standard scores, respectively, for HUIM-BPSO$^+$ and HUIM-BPSO$^-$ algorithms are both better than the HUPE$_{umu}$-GRAM$^-$ and HUPE$_{umu}$-GRAM$^+$ algorithms in terms of the number of the discovered HUIs.

From the above results, it can be also observed that the HUPE$_{umu}$-GRAM$^+$ algorithm performs sometimes worse than the HUPE$_{umu}$-GRAM$^-$ algorithm. This is reasonable since the designed OR/NOR-tree can only be used in the initial phase of the HUPE$_{umu}$-GRAM$^+$ algorithm. The main procedure of those two algorithms is still the same by performing the crossover and mutation operations to randomly generate the next solutions in the evolution process. Thus, the designed OR/NOR-tree is insufficient to improve the performance the GA-based approaches.

Table 10 Percentages of discovered HUIs of the GA-based approaches

Algorithm		$HUPE_{umu}$ -$GRAM^{+}$			$HUPE_{umu}$-$GRAM^{-}$		
Dataset	δ (%)	Best (%)	Average (%)	Worst (%)	Best (%)	Average (%)	Worst (%)
Chess	25 %	16.32	14.10	11.22	15.41	14.06	12.24
	25.5	26.82	25.39	21.95	26.82	22.05	19.51
	26%	37.50	25.5	18.75	31.25	18.95	12.50
	26.5	40	22	20	40	21.5	0
	27	0	0	0	0	0	0
Mushroom	14	16.41	13.78	11.94	14.92	12.01	10.44
	14.25	21.05	16.02	10.52	18.42	13.25	10.52
	14.5	15.78	11.15	5.26	15.78	11.01	0
	14.75	0	0	0	0	0	0
	15	0	0	0	0	0	0
Connect	28.9	18.51	14.83	12.34	17.28	14.81	13.58
	29.1	25	23.5	17.5	25	22.6	17.5
	29.3	25	16.2	10	30	19	15
	29.5	25	12.6	12.5	12.5	5.6	0
	29.7	0	0	0	0	0	0
Accidents	13.1	8.51	7.54	7.44	9.57	7.34	6.38
	13.4	8.62	6.89	6.89	6.91	5.17	5.11
	13.7	5.71	2.85	2.93	2.85	2.85	2.85
	14	0	0	0	0	0	0
	14.3	0	0	0	0	0	0
Foodmart	0.11	0	0	0	0	0	0
	0.12	0	0	0	0	0	0
	0.13	0	0	0	0	0	0
	0.14	0	0	0	0	0	0
	0.15	0	0	0	0	0	0
Retail	0.3	0	0	0	0	0	0
	0.4	0	0	0	0	0	0
	0.5	0	0	0	0	0	0
	0.6	0	0	0	0	0	0
	0.7	0	0	0	0	0	0

6.3 Number of determined patterns

In this section, the number of HUIs, 1-HTWUIs, and maximal patterns of different dataset w.r.t. variants of minimum utility thresholds is evaluated. The results are shown in Fig. 10.

From Fig. 10a–d it can be seen that the difference between the number of HUIs and the number of 1-HTWUIs is very large in all datasets. For the HUIM-$BPSO^{-}$ algorithm, the valid HUIs are retried from the permutations of 1-HTWUIs. This process takes all possible combinations and some of them may not exist in the database, especially in the sparse one. For the developed HUIM-$BPSO^{+}$ algorithm, the valid HUIs are determined by the designed OR/NOR-tree, which can greatly reduce the number of invalid (not existing in database) combinations. Thus, the proposed HUIM-$BPSO^{+}$ algorithm has better results than HUIM-$BPSO^{-}$ algorithm. From Fig. 10e–f, it is also found that the difference between the number of HUIs and the number of 1-HTWUIs in sparse dataset is also very large. The same results can be also observed from the difference between the number of HUIs and the number of maximal patterns. For example, the number of HUIs, the number of 1-HTWUIs, and the number of maximal patterns of chess dataset with minimum support threshold 26 % are, respectively, 16, 51, and 1409. It indicates that the size of chromosome is set as 51 in the designed algorithm. For the developed HUIM-$BPSO^{+}$ algorithm, the actual HUIs from 1409 patterns of the OR/NOR-tree need to be found. The other algorithms have to, however, discover the actual HUIs from the combinational process of 51 patterns, which require more computations than that of the HUIM-$BPSO^{+}$ algorithm. From the observed results in Fig. 10, we can conclude that the evolution process of the designed

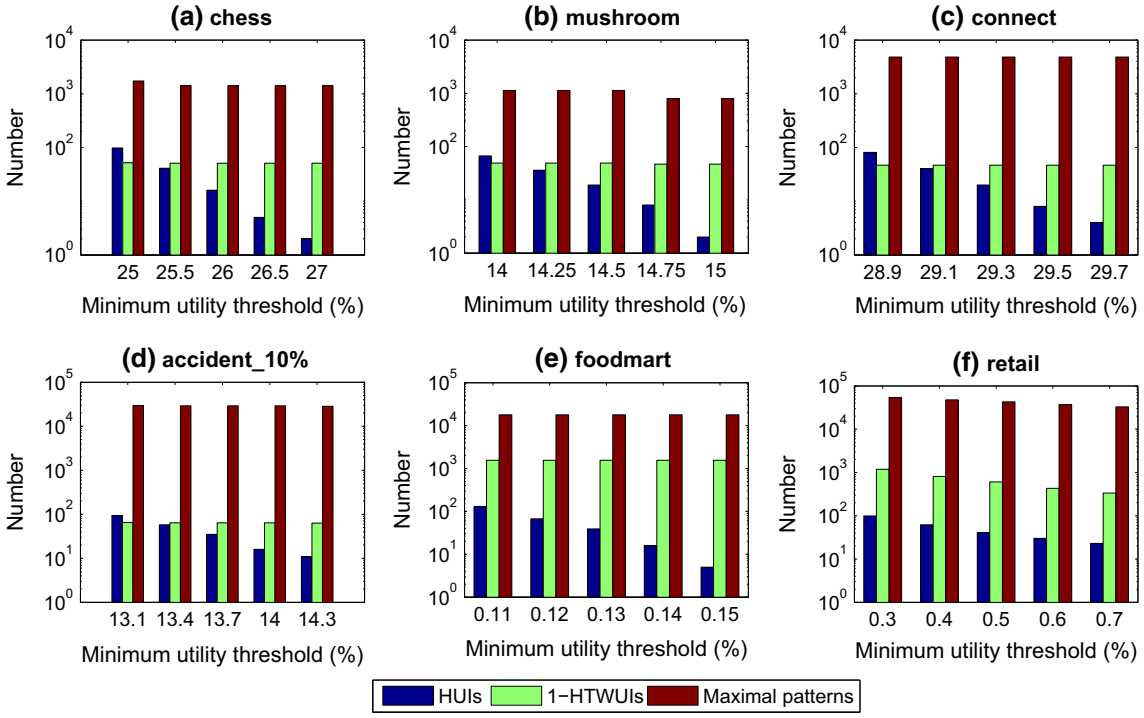

Fig. 10 Number of determined patterns w.r.t. variants of minimum utility thresholds

HUIM-BPSO$^+$ algorithm can efficiently find the actual HUIs from very large combinations, and the designed HUIM-BPSO$^+$ algorithm can limit the search space for retrieving the valid itemsets from the maximal patterns, thus speeding up the computations than the HUIM-BPSO$^-$ algorithm and the other two GA-based algorithms.

6.4 Convergence

In this section, the convergence of the four algorithms are evaluated in different datasets. The results are shown in Fig. 11.

From Fig. 11, it can be observed that the speed of convergence of the improved HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ are both slower than that of the designed HUIM-BPSO$^+$ algorithm, and the developed HUIM-BPSO$^+$ algorithm has better performance than the HUIM-BPSO$^-$ algorithm. The reason is that the HUIM-BPSO$^+$ algorithm adopts the developed OR/NOR-tree structure to avoid the invalid combinations; the generated itemsets of the HUIM-BPSO$^+$ algorithm can be concerned as the highly potential HUIs. From the conducted results shown in Fig. 11a, c, it can be found that the HUIM-BPSO$^+$ algorithm can efficiently discover the same number of HUIs as the TWU model. From Fig. 11b, d, the HUIM-BPSO$^+$ algorithm still has better results than the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms and the number of discovered HUIs is steadily increased along with the number of iterations. For a very sparse foodmart dataset, the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms generate, however, no HUIs since the number of 1-HTWUIs is very high but only fewer HUIs can be discovered from a very sparse dataset. Both the HUPE$_{umu}$-GRAM$^+$ and HUPE$_{umu}$-GRAM$^-$ algorithms suffer the combinational explosion in the evolution process. The proposed HUIM-BPSO$^+$ algorithm still generates the HUIs and is steadily increased along with the number of iterations. Overall, the proposed HUIM-PSO$^+$ algorithm has better convergence than the other algorithms.

7 Conclusions

High-utility itemset mining (HUIM) is emerging as an important topic in recent years since it can reveal the highly profitable products instead of the frequent itemset mining (FIM). Several algorithms have been proposed to efficiently mine the high-utility itemsets (HUIs) from the quantitative databases and most of them applied the statistical analysis to discover the required information. This process may take very large computations when the number of distinct items or the size of the database is very large. In the past, a GA-based approach, namely HUPE$_{umu}$-GRAM, was proposed to mine HUIs based on genetic algorithm. It suffers the combinational problem of the generated itemsets and requires several parameters in the evolution process.

In this paper, the particle swarm optimization (PSO)-based HUIM-BPSO algorithm is proposed to efficiently

Table 11 Standard scores of the discovered HUIs

Algorithm		HUIM-BPSO$^+$	HUIM-BPSO$^-$	HUPE$_{umu}$-GRAM$^+$	HUPE$_{umu}$-GRAM$^-$
Dataset	δ (%)	Standard score	Standard score	Standard score	Standard score
Chess	25	1.0405	0.6715	−0.8561	−0.8561
	25.5	0.9786	0.7449	−0.8326	−0.8910
	26	0.8646	0.8646	−0.7954	−0.9338
	26.5	0.8660	0.8660	−0.8660	−0.8660
	27	0.8660	0.8660	−0.8660	−0.8660
Mushroom	14	1.1928	0.4587	−0.7891	−0.8625
	14.25	1.2938	0.2812	−0.7313	−0.8438
	14.5	1.3754	0.1058	−0.7406	−0.7406
	14.75	1.3055	0.2611	−0.7833	−0.7833
	15	1.3055	0.2611	−0.7833	−0.7833
Connect	28.9	1.0056	0.7141	−0.8598	−0.8598
	29.1	0.9847	0.7385	−0.8616	−0.8616
	29.3	0.8652	0.8652	−0.9176	−0.8127
	29.5	0.8622	0.8622	−0.7472	−0.9771
	29.7	0.8660	0.8660	−0.8660	−0.8660
Accidents	13.1	1.1449	0.5319	−0.8384	−0.8384
	13.4	1.1063	0.5857	−0.8200	−0.8720
	13.7	1.1768	0.4845	−0.8307	−0.8307
	14	1.1066	0.5858	−0.8462	−0.8462
	14.3	1.2032	0.4433	−0.8232	−0.8232
Foodmart	0.11	1.5	−0.5	−0.5	−0.5
	0.12	1.5	−0.5	−0.5	−0.5
	0.13	1.5	−0.5	−0.5	−0.5
	0.14	1.5	−0.5	−0.5	−0.5
	0.15	1.5	−0.5	−0.5	−0.5
Retail	0.3	1.5	−0.5	−0.5	−0.5
	0.4	1.5	−0.5	−0.5	−0.5
	0.5	1.5	−0.5	−0.5	−0.5
	0.6	1.5	−0.5	−0.5	−0.5
	0.7	1.5	−0.5	−0.5	−0.5

mine HUIs. The designed algorithm first adopts the TWU model to find the number of high-transaction-weighted utilization 1-itemsets (1-HTWUIs) as the particle size, which can greatly reduce the combinational problem in the evolution process. The sigmoid function of the discrete (binary) PSO (BPSO) is adopted to mine the HUIs in the evolution process. An OR/NOR-tree structure is further designed to avoid the invalid combinations of the particles, thus speeding up the computations for discovering the HUIs. From the conducted experiments, the proposed HUIM-BPSO algorithm has better results than the GA-based algorithms in terms of execution time, the ability for discovering HUIs, and the convergence.

Up to now, only few works were proposed to mine high-utility itemsets based on evolutionary computation. In this paper, we address the issue of HUIM using the binary PSO mechanism with the developed OR/NOR-tree structure to efficiently mine the HUIs. The reason is that the PSO-based approach can be easily implemented without the trivial task to set the appropriate crossover and mutation rates as the GA-based algorithm does. The other optimization algorithms such as artificial immune system, ant colony optimization, and artificial bee colony mechanism could also be adopted and modified to efficiently mine the HUIs. However, it is a non-trivial task since some algorithms can only be used to handle the continuous problem and some algorithms perform the operations with randomization; the designed algorithm cannot be directly applied to handle the issue of HUIM. More mechanisms can be further studied as our future works.

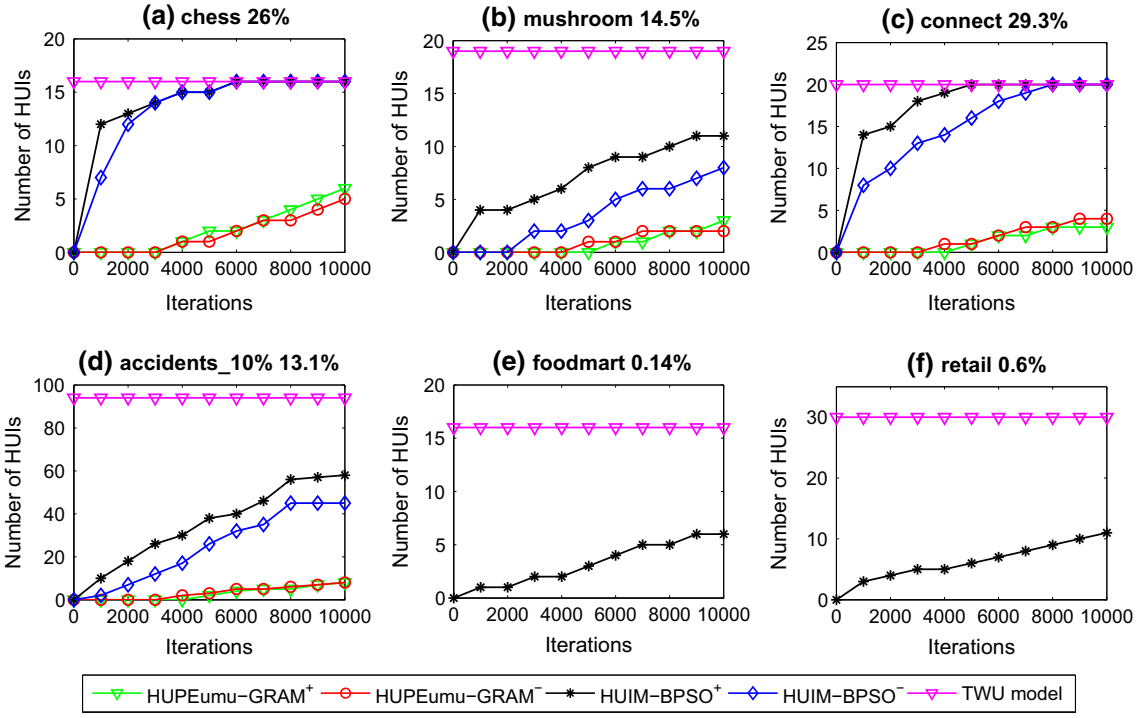

Fig. 11 Convergence w.r.t variants of iterations

Acknowledgments This research was partially supported by the Shenzhen Peacock Project, China, under Grant KQC201109020055A, by the National Natural Science Foundation of China (NSFC) under Grant No. 61503092, by the Natural Scientific Research Innovation Foundation in Harbin Institute of Technology under Grant HIT.NSRIF. 2014100, and by the Shenzhen Strategic Emerging Industries Program under Grant ZDSY20120613125016389.

Compliance with ethical standards

Conflict of interest The authors declare that there are no conflicts of interest in this paper.

Ethical approval This article does not contain any studies with human participants performed by any of the authors.

References

Agrawal S, Silakari S (2013) FRPSO: Fletcher-Reeves based particle swarm optimization for multimodal function optimization. Soft Comput 18(11):2227–2243

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. Int Conf Very Large Data Bases 1215:487–499

Ahmed CF, Tanbeer SK, Jeong BS, Le YK (2009) Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans Knowl Data Eng 21(12):1708–1721

Cattral R, Oppacher F, Graham KJL (2009) Techniques for evolutionary rule discovery in data mining. IEEE Congr Evolut Comput :1737–1744

Chan R, Yang Q, Shen YD (2003) Minging high utility itemsets. IEEE Int Conf Data Mining :19–26

Chen MS, Han J, Yu PS (1996) Data mining: an overview from a database perspective. IEEE Trans Knowl Data Eng 8(6):866–883

Fournier-Viger P, Wu CW, Zida S, Tseng VS (2014) FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. Found Intell Syst 8502:83–92

Fournier-Viger P, Wu CW, Tseng VS (2014) Novel concise representations of high utility itemsets using generator patterns. Adv Data Mining Appl 8933:30–43

Fournier-Viger P, Zida S (2015) FOSHU: faster on-shelf high utility itemsets mining with or without negative unit profit. ACM Symp Appl Comput :857–864

Frequent itemset mining dataset repository (2012). http://fimi.ua.ac.be/data/

Gong W, Cai Z, Ling CX (2010) DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization. Soft Comput 15(4):645–665

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Disc 8(1):53–87

Holland J (1975) Adaptation in Natural and Artificial Systems, Cambridge. MIT Press, USA

Kannimuthu S, Premalatha K (2014) Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl Artif Intell 28(4):337–359

Kennedy J, Eberhart R (1997) A discrete binary version of particle swarm algorithm. IEEE Int Conf Syst Man Cybern 5:4104–4108

Kennedy J, Eberhart R (1995) Particle swarm optimization. IEEE Int Conf Neural Netw 4:1942–1948

Kuo RJ, Chao CM, Chiu YT (2011) Application of particle swarm optimization to association rule mining. Appl Soft Comput 11(1):326336

Lan GC, Hong TP, Tseng VS (2013) An efficient projection-based indexing approach for mining high utility itemsets. Knowl Inf Syst 38(1):85–107

Li XT, Yin MH (2015) A particle swarm inspired cuckoo search algorithm for real parameter optimization. Soft Comput :1–25

Liang XL, Li WF, Zhang Y, Zhou MC (2014) An adaptive particle swarm optimization method based on clustering. Soft Comput 19(2):431–448

Lin CW, Gan WS, Fournier-Viger P, Hong TP (2015) Mining high-utility itemsets with multiple minimum utility thresholds. Int C* Conf Comput Sci Softw Eng :9–17

Lin JCW, Yang L, Fournier-Viger P, Wu MT, Hong TP, Wang LSL (2015) A Swarm-based approach to mine high-utility itemsets. Multidiscip Int Soc Netw Conf

Lin CW, Hong TP, Lu WH (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419–7424

Liu Y, Liao WK, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. Lecture Notes Comput Sci :689–695

Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. ACM Int Conf Inf Knowl Manag :55–64

Martnez-Ballesteros M, Martnez-lvarez F, Riquelme JC (2010) Mining quantitative association rules based on evolutionary computation and its application to atmospheric pollution. Integr Comput Aided Eng 17(3):227–242

Menhas MI, Fei M, Wang L, Fu X (2011) A novel hybrid binary PSO algorithm. Lect Notes Comput Sci 6728:93–100

Microsoft (1996) Example database foodmart of Microsoft analysis services. http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx

Nouaouria N, Boukadouma M, Proulx R (2013) Particle swarm classification: a survey and positioning. Pattern Recogn 46(7):20282044

Pears R, Koh YS (2012) Weighted association rule mining using particle swarm pptimization. Lect Notes Comput Sci 7104:327–338

Salleb-Aouissi A, Vrain C, Nortet C (2007) QuantMiner: a genetic algorithm for mining quantitative association rules. Int Jt Conf Artif Intell 7:1035–1040

Sarath KNVD, Ravi V (2013) Association rule mining using binary particle swarm optimization. Eng Appl Artif Intell 26:1832–1840

Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359

Tsai CW, Huang KW, Yang CS, Chiang MC (2015) A fast particle swarm optimization for clustering. Soft Comput 19(2):321–338

Tseng VS, Wu CW, Shie BE, Yu PS (2010) UP-growth: an efficient algorithm for high utility itemset mining. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 253–262

Wu CW, Shie BE, Tseng VS, Yu PS (2012) Mining top-k high utility itemsets. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 78–86

Yao H, Hamilton HJ, Butz CJ (2004) A foundational approach to mining itemset utilities from databases. SIAM Int Conf Data Mining 4:211–225

Yao H, Hamilton HJ (2006) Mining itemset utilities from transaction databases. Data Knowl Eng 59(3):603–626

Yen SJ, Lee YS (2007) Mining high utility quantitative association rules. Lect Notes Comput Sci 4654:283–292

Zida S, Fournier-Viger P, Lin CW, Wu CW, Tseng VS (2015) EFIM: a highly efficient algorithm for high-utility itemset mining. In: Mexican International Conference on Artificial Intelligence

Zihayat M, An A (2014) Mining top-k high utility patterns over data streams. Inf Sci 285:138–161