Journal of Systems Engineering and Electronics
Vol. 35, No. 4, August 2024, pp.965 – 975

PHUI-GA: GPU-based efficiency evolutionary algorithm for mining high utility itemsets

JIANG Haipeng[1], WU Guoqing[2,3], SUN Mengdan[2,3], LI Feng[2], SUN Yunfei[4], and FANG Wei[1,*]

1. Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Department of Computer Science and Technology, Jiangnan University, Wuxi 214122, China; 2. China Ship Scientific Research Center, Wuxi 214082, China; 3. Taihu Laboratory of Deepsea Technological Science, Wuxi 214082, China; 4. Department of Mathematics, Nanjing University, Nanjing 210023, China

Abstract: Evolutionary algorithms (EAs) have been used in high utility itemset mining (HUIM) to address the problem of discovering high utility itemsets (HUIs) in the exponential search space. EAs have good running and mining performance, but they still require huge computational resource and may miss many HUIs. Due to the good combination of EA and graphics processing unit (GPU), we propose a parallel genetic algorithm (GA) based on the platform of GPU for mining HUIM (PHUI-GA). The evolution steps with improvements are performed in central processing unit (CPU) and the CPU intensive steps are sent to GPU to evaluate with multi-threaded processors. Experiments show that the mining performance of PHUI-GA outperforms the existing EAs. When mining 90% HUIs, the PHUI-GA is up to 188 times better than the existing EAs and up to 36 times better than the CPU parallel approach.

Keywords: high utility itemset mining (HUIM), graphics processing unit (GPU) parallel, genetic algorithm (GA), mining performance.

DOI: 10.23919/JSEE.2024.000020

1. Introduction

High utility itemset mining (HUIM) [1−3] is proposed to find the profitable products by considering both the profit and quantity factors instead of association-rule mining (ARM) [4] or frequent itemset mining (FIM) [5] which only considers the support mesure (frequency of the products). Large number of itemsets can be eliminated by the anti-monotonic in FIM, that is, an itemset cannot have a superset with a higher support value. The HUIM problem is widely recognized as more difficult than FIM because HUIM is not anti-monotonic [6]. The HUIM problem is intended to find the high-utility itemsets (HUIs) that the utility is higher than a given threshold [7]. HUIM has been widely used in real world, such as recommend system [8], stock marketing [9], and customer segmentation [10]. Some new topics on discovering HUIs have also been discussed in recent years, such as discovering the top-k HUIs [11], high average-utility itemsets [12], and high utility sequential patterns [13].

Various exact algorithms [14−17] have been proposed to solve the HUIM problems that can mine all the HUIs. However, the performance of these exact algorithms degrades sharply as the number of transactions and distinct items increases due to the enumerate of large non-existent itemsets [18]. Some spark based CPU parallel algorithms [19−21] are proposed to improve the mining performance, but the algorithms still face the problem of huge calculation in a progressive structure when the number of items becomes larger. Evolutionary algorithms (EAs) are applied in HUIM to solve the performance bottleneck of exact algorithms, including genetic algorithm (GA) [22], particle swarm optimization (PSO) [23,24], bat algorithm (BA) [25,26], and ant colony optimization (ACO) [27]. In addition, fish swarm algorithm and simulated annealing [18,28] are also used in HUIM. However, the distribution of HUIs is not even, only the best value is recorded in these traditional EAs leading to deficiency of solutions. Therefore, several strategies such as roulette wheel selection and promising encoding vector (PEV) check are applied in [29,30] to accelerate the convergence speed. However, the existing strategy has difficulty in mining the HUIs in the first few iterations, which wastes iteration performance. What's more, the fitness evaluation step and PEV strategies are time consuming in CPU. Although the EAs are faster than the traditional

Manuscript received December 21, 2022.
*Corresponding author.
This work was supported by the National Natural Science Foundation of China(62073155;62002137;62106088;62206113), the High-End Foreign Expert Recruitment Plan (G2023144007L), and the Fundamental Research Funds for the Central Universities(JUSRP221028).

exact algorithms, they are still slower than the CPU parallel exact approaches. The GA is suitable for GPU parallel. However, some existing strategies run serial in CPU such as PEV check, which may not perform well on GPU directly. Besides, roulette wheel selection is a serial process and is difficult to run more quickly on GPU than on CPU. Therefore, design strategies running on CPU and on GPU to consider both running speed and the mining quality is challenging.

In this paper, we propose a GPU parallel GA for mining HUIM (PHUI-GA) in a shorter time and keep good mining quality. To the best of our knowledge, PHUI-GA is the only work which introduces EA for HUIM on GPU. Three strategies are proposed on origin GA, which speeds up the algorithm through mining performance. The CPU-time intensive steps of the algorithm are performed on GPU to reduce the computational time in parallel. The main contributions are summarized as follows:

(i) We design an improved GA on a CPU/GPU architecture following a master/worker model. CPU is the master that decomposes the task after improved GA steps and distributes them to different threads on GPU, the threads are the workers. The iterative strategies to improve the convergence speed is run on CPU and the CPU intensive tasks are performed on GPU until the fitness value is calculated. Different parallel designs are adopted on the GPU according to the characteristic of the strategies.

(ii) Under the proposed GPU framework, three strategies are proposed to improve the convergence speed, which are the improved population initialization strategy with the use of support measure, a flexible promising encoding vector checking (FPEVC) to convert non-existent itemsets into reasonable itemsets, and a re-exploration strategy to make better use of the HUIs that have been mined.

(iii) The experiments show that, PHUI-GA outperforms existing EAs on convergence speed and mining quality in same iterations. The parallel on GPU is valid that some strategies are accelerated up to 250 times (250×) than strategies before parallel. When mining 90% HUIs, the running speed compared with EAs and the CPU parallel exact algorithm is remarkable.

The rest of this paper is as follows. Related work of HUIM is introduced in Section 2. The HUIM problem is described in Section 3. The proposed PHUI-GA framework is presented in Section 4. Experimental results on four public datasets are presented and analyzed in Section 5. Finally, conclusions and future work are drawn in Section 6.

2. Related works

Current HUIM algorithms can be categorized into two types, which are exact HUIM algorithms and EAs. As exact HUIM algorithms are considered, such as Up-Growth [31] and HUI-Miner [15], the candidate itemsets are generated in the first phase and the verification is done in the second phase. Up-Growth [31] builds FP-tree to reduce the overestimated utilities. The utility-lists and a level-wise algorithm are proposed in HUI-Miner [15] for efficiently mining HUIs. However, the exact HUIM algorithms face the problem of high time complexity when enumerating all the items. For the purpose of improving the running performance of exact HUIM (HUIM BPSO) algorithm with CPU parallel, Chen et al. introduced methods based on the map-reduce model [19−21]. For example, PHUI-Miner [19] and parallel faster high utility itemset mining (PFHM) [20] are proposed to directly parallelize HUI-Miner and faster high utility itemset mining (FHM). However, the level-wise strategy in these algorithms increases the amount of computation exponentially, and maintaining a large number of utility-lists in memory for a long time is costly.

A set of EAs are proposed for HUIM to solve the performance bottleneck of the exact algorithms, including GA, PSO and BA. High utility pattern extraction using GA with ranked mutation using minimum utility threshold (HUPEUMU-GARM) and high utility pattern extraction using GA with ranked mutation without using minimum utility threshold (HUPEWUMU-GARM) [22] use GA to solve the HUIM problem, the difference between the two algorithms is that HUPEWUMU-GARM does not require the minimum threshold. PSO algorithms are used in HUIM, which are HUIM based on binary PSO (HUIM-BPSO) algorithm [23] and HUIM-BPSO-Tree [24]. HUIM-BPSO outperforms the above EAs with the use of an OR/NOR-tree structure. High utility itemset mining framework (HUIF) is proposed [29] to improve the diversity of solutions in limited iterations. Several techniques such as Repair are used in improved GA for HUIM (HUIM-IGA) [30] to further accelerate the convergence speed. Recently, a hill climbing method based on simulated annealing (SA) [18] is proposed for mining more HUIs in low threshold. A set-based PSO (SPSO) [32] is proposed with a cut-set structure to maintain positions with a higher running speed than traditional PSO. The artificial fish swarm algorithm (AFSA) [28] is also applied for HUIM to avoid results distributed around a few extreme points. Although EAs have gained good computational effects, the fitness evaluation step and PEV based strategies are still time consuming.

Some GPU-based parallel EAs are proposed in

[33−35], however, they are only designed for FIM but not in HUIM. Single evaluation in GPU (SE-GPU) [35] parallelizes the fitness evaluation step to improve running speed. All bees swarm optimization (BSO) steps in GPU-based BSO miner (GBSO-Miner) [34] are performed on GPU to further accelerate the FIM problem. GA that runs on clusters of GPUs in cluster GPU GA (CGPUGA) [33] is to discover diversified FIM. However, mining FIM is easier than HUIM so that the same strategies and the acceleration on FIM cannot be used in HUIM to find HUIs. As EAs are considered, GPU parallelism has not been studied enough to improve the computational performance in HUIM.

3. Preliminary

Let $D = \{T_1, T_2, \cdots, T_n\}$ be a transaction dataset with n transactions. T_n ($1 \leqslant n \leqslant N$) represents a transaction in the dataset. Let $I = \{i_1, i_2, \cdots, i_p\}$ be a finite set of items. In D, each T_n contains multiple unique items in I.

The utility of item i_p in transaction T_n is denoted as $u(i_p, T_n) = p(i_p, T_n) \cdot q(i_p)$, in which $p(i_p, T_n)$ represents the quantity of item i_p in T_n, $q(i_p)$ represents the utility of item. Let an itemset be X. The utility of itemset X in transaction T_n is defined as $u(X, T_n) = \sum_{i_p \in X \wedge X \subseteq T_n} u(i_p, T_n)$ and the utility of itemset X in dataset is defined as $u(X) = \sum_{X \subseteq T_n \wedge T_n \in D} u(X, T_n)$. The transaction utility of transaction T_n is defined as $\mathrm{TU}(T_n) = \sum_{i \in T_n} u(i_p, T_n)$. The minimum utility threshold is defined as $\mathrm{minUtility} = \delta \cdot \sum_{T_n \in \mathbf{D}} \mathrm{TU}(T_n)$ where δ is a user defined percentage of the total TU value of the dataset. The transaction-weighted utilization (TWU) of an itemset X is defined as $\mathrm{TWU}(X) = \sum_{X \subseteq T_n \wedge T_n \in D} \mathrm{TU}(T_n)$.

There is a pruning strategy. We can define the itemset X as a high transaction-weighted utilization itemset (HTWUI) if $\mathrm{TWU}(X) \geqslant \mathrm{minUtility}$; otherwise, X is a low transaction-weighted utilization itemset (LTWUI). An HTWUI/LTWUI with i items is called i-HTWUI/i-LTWUI. If $\mathrm{TWU}(X) < \mathrm{minUtility}$, then its superset must be i-LTWUI and can be pruned because the set of all HUIs is a subset of the set of HTWUIs and no LTWUI is an HUI [18].

A running example is considered in Table 1 and the utility of each item is shown in Table 2. Let an itemset X be $\{b, e\}$, then the utility of X is $u(X) = u(X, T_3) + u(X, T_5) = u(b, T_3) + u(b, T_5) + u(e, T_3) + u(\mathrm{e}, T_5) = 1 \times 2 + 1 \times 2 + 1 \times 2 + 1 \times 2 = 8$. If the given minimum threshold minUtility = 9, then $\{b, e\}$ is not an HUI.

Table 1 HUI

D	Transaction	TU
T_1	$(a, 1)$ $(b, 1)$	5
T_2	$(c, 2)$	2
T_3	$(b, 1)$, $(e, 1)$, $(f, 1)$	9
T_4	$(d, 3)$, $(e, 1)$	14
T_5	$(a, 1)$, $(b, 1)$, $(c, 1)$, $(d, 1)$, $(e, 1)$, $(f, 1)$	17
T_6	$(a, 4)$, $(c, 1)$	13

Table 2 Utility values of items

Item	Utility
a	3
b	2
c	1
d	4
e	2
f	5

For the pruning strategy of 1-HTWUI/1-LTWUI, $\mathrm{TWU}(\{b\}) = \mathrm{TU}(T_1) + \mathrm{TU}(T_3) + \mathrm{TU}(T_5) = u(b, T_1) + u(b, T_3) + u(b, T_5) = 5 + 9 + 17 = 31$. If minUtility = 30, then TWU(b) is a 1-HTWUI and the superset of $\{b\}$ can be further evaluated, but $\mathrm{TWU}(f) = 9 + 17 = 26 < 30$ and its superset can be pruned.

4. The proposed algorithm

4.1 Framework of PHUI-GA

Fig. 1 shows the overall structure of PHUI-GA framework. The master is run on CPU and the slave is offloaded to run on GPU. The evolution steps are run on CPU and the GPU receives tasks from the master to run time consuming steps until fitness of each individual is obtained.

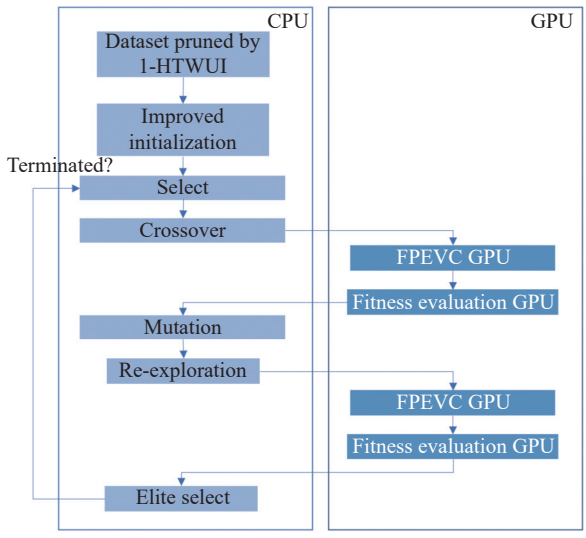


Fig. 1 Framework of PHUI-GA

First, the dataset is scanned twice to prune the items by 1-HTWUI mentioned in Section 3. Then initialize the population with the improved initialization strategy. In the iteration steps, the GA steps (the roulette select, uniform crossover, mutation, and elite selection) are performed on CPU. Re-exploration is executed after mutation but before elite selection. After crossover and re-exploration, FPEVC and fitness evaluation step are executed on GPU platform to get fitness of each individual in the population. After the fitness is obtained, the itemsets are stored as HUIs if the fitness value of the individual is higher than the minimum threshold.

FPEVC has a serial structure, therefore the time consuming steps runs on GPU and CPU is used for synchronization. Fitness evaluation step of each individual is performed the same time on GPU. The combination of the CPU/GPU process is repeated until enough HUIs are mined or the maximum iteration is reached.

4.2 Algorithm improvements

4.2.1 Improved initialization strategy with support values

Binary encoding is usually used in EAs, where 1 means there exists an item and 0 means the item does not exist. For example, {110010} represents itemset X is $\{a, b, e\}$. The existing initial strategies are difficult to generate HUIs in the early iterations.

Therefore, first we only use half items sorted by TWU to initialize k items by roulette selection. Then, another top-m high support items are set to 1 according to a random number $\alpha(0 < \alpha < 1)$ if $\alpha > 0.5$, which means several high frequency items are used. We find that the item with the highest frequency is always the high probability items in HUIM problems. As shown in Fig. 2, for example, if $k = 2$ and $m = 1$. Half items sorted by TWU is f, d, a. Let the itemset X of an individual after k item roulette selection be {001010}, representing itemset $\{c, e\}$ and $u(\{c, e\}) = 3$. If the highest frequency item is b and $\delta = 0.75$, then b is also initialed, then X is $\{a, b, d\}$ and $u(\{b, c, e\}) = 5$. $u(\{b, c, e\})$ is higher than $u(\{c, e\})$.

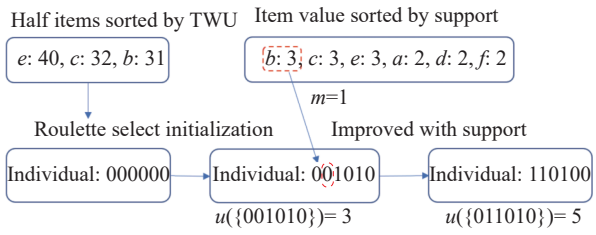


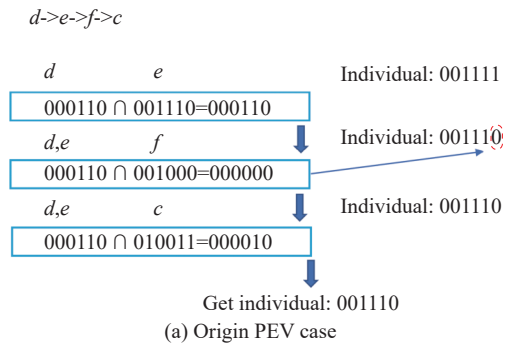
Fig. 2 Improved initialization

4.2.2 FPEVC strategy

Each solution of an individual is a potential HUI, and is represented as an encoding vector (EV) [18]. Let Bit(i_p) represent the bitmap cover of item i_p and Bit(X) represent the bitmap cover of itemset X. If Bit(X) only contains 0 s then EV is an unpromising encoding vector (UPEV), otherwise EV is a PEV. However, existing PEV based strategy only prunes from the single direction, which results in that a large number of repeated PEVs are generated in each iteration step.

To provide more processing direction, an FPEVC strategy is proposed. For FPEVC, the sort order of TWU is the processing direction and then two random positions in the processing direction are selected and swapped, which result in a flexible processing direction. Thus first initialize the bitmap cover of the first item according to the flexible processing direction, then a bitwise-AND operation with the bitmap cover of the next item is done. If the result is UPEV, then trace back to the step before bitwise-AND operation. If the result is PEV, then the next bitwise-AND operation is done. The strategy executed until all the bitmap covers of items are traversed.

As shown in Fig. 3, if current individual is {001111}, with $u(c) = 4$, $u(d) = 16$, $u(e) = 6$, $u(f) = 5$, the origin processing direction is d, e, f, c. Next Bit(d) ∩ Bit(e) = {000110} ∩{001110} = {000110}. Bit({d, e}) is a PEV, then Bit({d, e}) ∩ Bit(f) = {000110} ∩{001000} = {000000} is the UPEV, f is removed. Then Bit({d, e}) ∩ Bit(c)= {000110} ∩ {010011} = {000010}. With this operator, the itemset after processed is $\{c, d, e\}$ in T_5 which is a PEV. If a same individual {001111} is searched in another FPEVC, the pruning order of d and f may be swapped. Then the individual $\{e, f\}$ is got, which is different from $\{c, d, e\}$. Due to the different pruning directions, the same individual can be processed by the FPEVC strategy to obtain the different promising itemsets during the iteration, which provide more possibilities in iteration steps.



(a) Origin PEV case

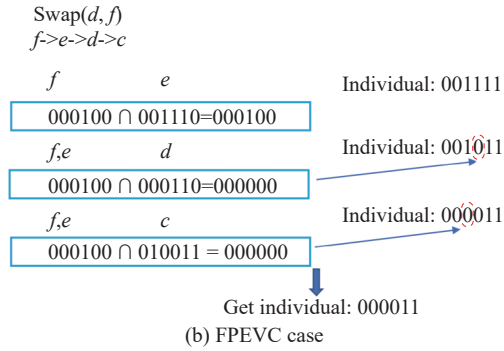


Fig. 3 Two different pruning directions with FPEVC in a same individual

4.2.3 Re-exploration strategy

Some itemsets of individuals in the iteration may have existed in the discovered HUIs, which wastes the computational resource since fitness values are evaluated repeatedly. Therefore, the re-exploration strategy is proposed to make better use of these repeated itemsets.

First, the discovered HUIs are stored in a hash table. If the itemset X exists in the hash table, the re-exploration strategy is executed. In the re-exploration strategy, two random positions of the searched individual are flipped to increase the opportunity for individuals to explore new search areas, which make the discovered repeated individuals be fully used. For example, as shown in Fig. 4, if X is {101 000} and already exists in the hash table, the re-exploration strategy is performed to generate two random positions r_1= 3 and r_2 = 5. Then the 3rd position of the individual is flipped from 1 to 0 and the 5th position of the individual is flipped from 0 to 1, and the new individual {100 010} is obtained. If X does not exist in the hash table, FPEVC and fitness evaluation steps are performed subsequently.

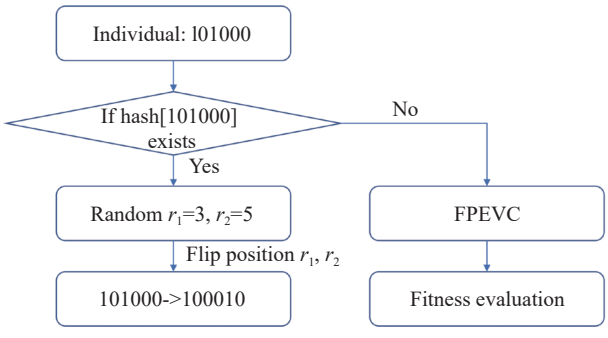


Fig. 4 Re-exploration strategy

4.3 Parallel design on GPU

4.3.1 FPEVC on GPU

As can be seen from Subsection 4.2.2, the FPEVC is a serial structure. However, the data transfer from trace back operation and the bitwise-AND operation when calculating PEV in each step is time consuming, which costs time when evaluated for many times. Therefore, GPU parallelism is used to reduce the time complexity when all the bitmap covers of items are traversed, CPU is used for synchronization. As shown in Algorithm 1, the processing direction is first calculated (Step 1−Step 2), then the result is initialized by the bitmap of the first item in the processing direction, Bit(i_1) is copied to a temporary bitmap on GPU (Step 3). Then each individual, which is recorded as an encoding vector EV, executes the main loop of FPEVC (Step 4−Step12). Step 5 performs the bitwise-AND operation in parallel. Then sum reduction is performed in Step 6 to calculate the sum 1-item in Step 5. If the sum number is 0, then change the position in EV to 0. The data transmission in middle steps are also executed on GPU (Step 8, Step 11).

Algorithm 1 FPEV-checking

Input: EV

Output: PEV

1: Let VN items be the number of 1-item in EV and denoted as $i_1, i_2, \cdots, i_{VN}$
2: Denote the FVN items be the VN items sorted with TWU and swap two item positions
3: tmp←Bit(i_1) on GPU
4: **for** k=2 to FVN **do**
5: mid←tmp∩Bit(i_k) on GPU
6: sum←Sum_reduction_GPU(mid)
7: **if** sum is 0 **then**
8: mid←tmp on GPU
9: change kth position in EV to 0
10: **else**
11: tmp←mid on GPU
12: **end if**
13: synchronization();
14: **end for**

4.3.2 Fitness evaluated step on GPU

Fitness evaluation is the most time-consuming step of EAs due to the huge computation on transactions in each iteration. We evaluate all the individual fitness simultaneously. In Algorithm 2, every block of threads matches with one transaction T_n in candidate list to get the utility of a transaction TU(T_n) and saved (Step 3−Step 4). Then, after the evaluation of each transaction of all the individuals (Step 5), a sum reduction operation (Step 6) [36] is used to get the final fitness.

Algorithm 2 Parallel calculate fitness

Input: N, dataset

Output: fitness

1: bid←blockIdx.y

2: d←treadIdx.x+blockDim.x×blockIdx.x;

3: tmp_fitness←calculate TU(T_d) of the transaction

4: fitness [bid×N+d]=tmp_fitness

5: synchronization();

6: Sum reduction(fitness)

4.3.3 Time complexity analysis of GPU based steps

In FPEVC, the time complexity of the sum reduction operation is reduced from $O(N)$ to $O(\log_2 N)$. The time complexity of each bitwise-AND operation is reduced from $O(N)$ to $O(1)$. The time complexity of bitmap assignment of the Bitmap is reduced from $O(N)$ to $O(1)$. The time complexity of FPEVC is $O(\log_2 N+1)$. Although many CPU synchronizations degrade parallel performance, the parallel is effective.

Let the maximum length of transaction be $\max_N$, and the population size be pop_size. The time complexity of the sum reduction operation is $O(\log_2 N)$. Therefore, the time complexity is reduced from $O(N\times \text{pop\_size}\times \max_N)$ to $O(\max_N+\log_2 N)$.

5. Experiments

5.1 Experimental settings

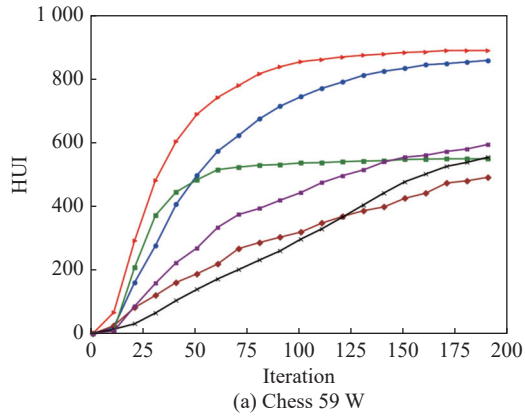
All the experiments are performed on a computer with a 16-Core 16 GB memory CPU and running on Windows 10. The GPU we use is RTX 3060. We evaluate the performance of PHUI-GA and compare it with five EAs, which are HUIM-IGA [30], HUIF-GA [30], HUIF-PSO [30], HUIF-BA [30], and HUIM-SA [18], and a CPU parallel algorithm PHUI-Miner [19]. Four datasets are used which are Chess, Mushroom, Connect and Accident. The characteristics of the datasets can be seen in Table 3. Accident is large so that we take 10% named Accident_10% [23]. Both compared algorithms and datasets are from an open-source data mining library. As for the parameters in the algorithms, both crossover and mutation rates in PHUI-GA are set to 0.5. The parameters k, m in the initialization strategy are set to 10, 5, respectively. The parameters of the comparison algorithms are consistent with those in the literature [10,11,13,16,22].

Table 3 Characteristics of the datasets

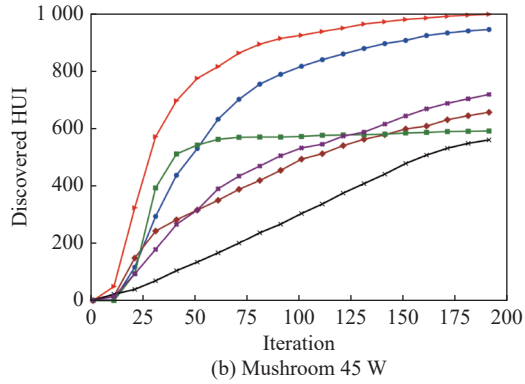
Dataset	Item	Transaction	Density/%
Chess	76	3 196	49.33
Mushroom	119	8 124	19.33
Connect	469	34 018	7.22
Accidents_10%	130	67 557	33.33

5.2 Comparison on convergence speed

The convergence speed of six different EAs algorithms on four datasets are compared here. The maximum number of iterations is 200 and the population size in each EA is 100. The minimum threshold of each dataset is shown at the top of the chart. For example, Chess 59 W means the minimum threshold in Chess dataset is 590 000. As shown in Fig. 5, HUIF-SA, HUIF-PSO, and HUIF-BA have linear convergence speed. Among the above three algorithms, HUIF-PSO obtains the best convergence speed. The mining performance of HUIM-SA is unstable due to the strong randomness. HUIF-GA is faster at first, but decreases after a few iterative steps. The convergence speed of HUIM-IGA is better than HUIF-GA with the number of iterations. With the use of improved initialization strategy, PHUI-GA can mine more HUIs at first, and with another two improvements, the mining performance of each iteration is better than all the EAs. With the improvements, the convergence performance of PHUI-GA is better than all the contrast algorithms.



(a) Chess 59 W



(b) Mushroom 45 W

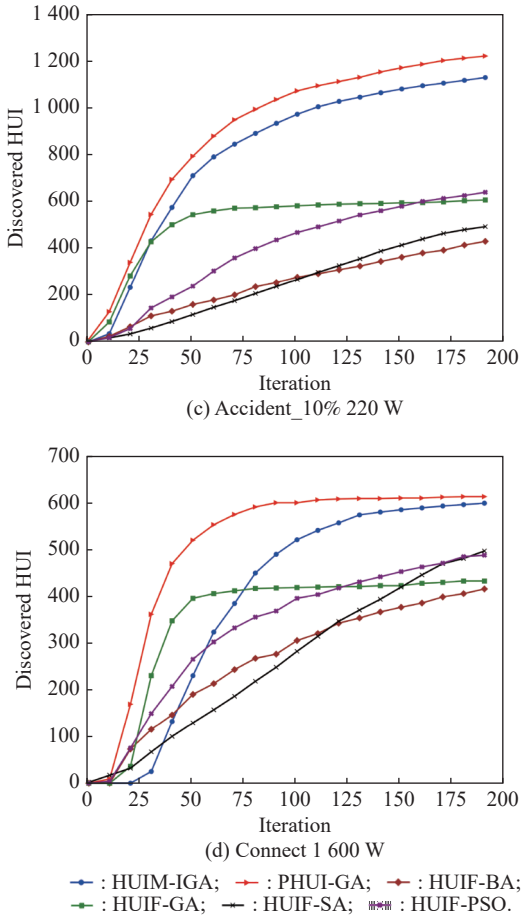


Fig. 5 Mining performance comparison on convergence speed

5.3 Performance comparison on mining quality

The mining quality of six algorithms are compared in this section. Five different minimum thresholds are used in each dataset. For example, the minimum thresholds used in Chess are 56 W, 57 W, 58 W, 59 W, and 60 W. As shown in Fig. 6. The mining performance of HUIF-PSO is better than HUIF-GA and HUIF-BA because HUIF-GA falls into local optimal and HUIF-BA lacks population diversity strategy. HUIM-SA has poor mining quality in Chess and Mushroom, but in Accident_10% and Connect, HUIM-SA can mine more HUIs than HUIF-BA. The mining quality of HUIM-IGA is better than HUIF-PSO, HUIF-GA, HUIF-BA, and HUIF-SA. On the threshold of 1 560 W on Connect, the mining quality of HUIM-IGA can be twice better than HUIF-GA. The mining quality of PHUI-GA is higher than HUIM-IGA on all the datasets, and the improvement of mining quality compared with PHUI-GA is 5%−20% higher than that of HUIM-IGA on average, the improvement of mining quality becomes more obvious as the minimum threshold decreases. Compared with other four algorithms, the mining quality in same iterations can reach about 50% improvement.

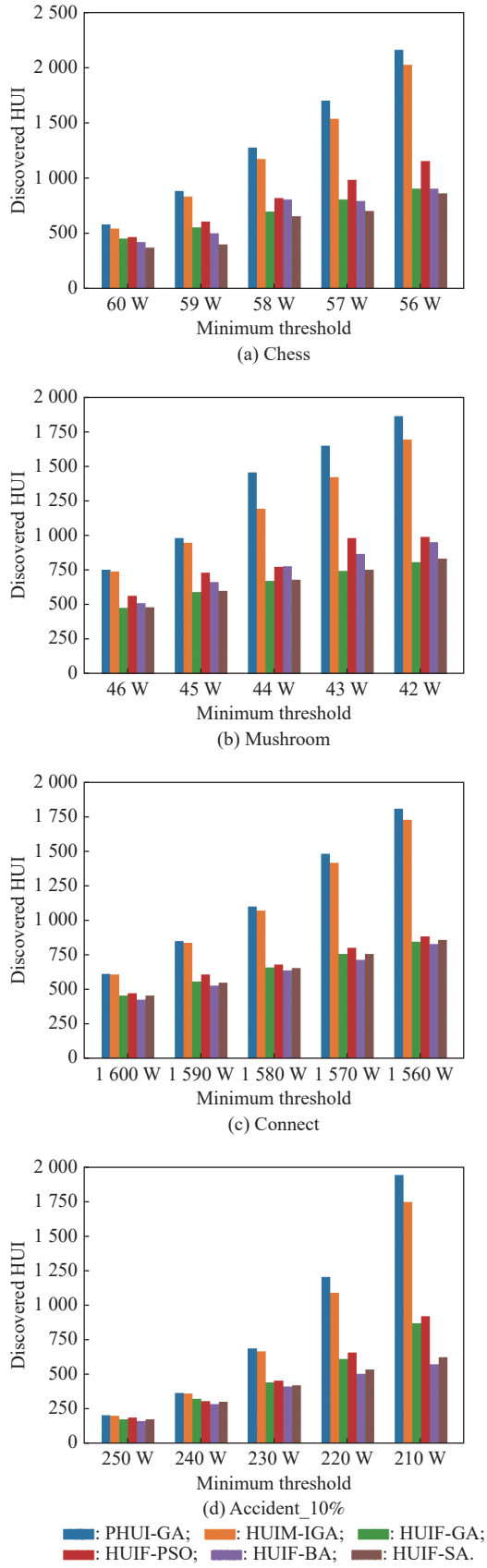


Fig. 6 Comparison on mining quality

5.4 Parallel performance investigation

In this section, the parallel performance of the iterative step, fitness evaluation step and FPEVC are shown in Fig. 7. The blue and red bars represent the running time before and after GPU parallel, respectively.

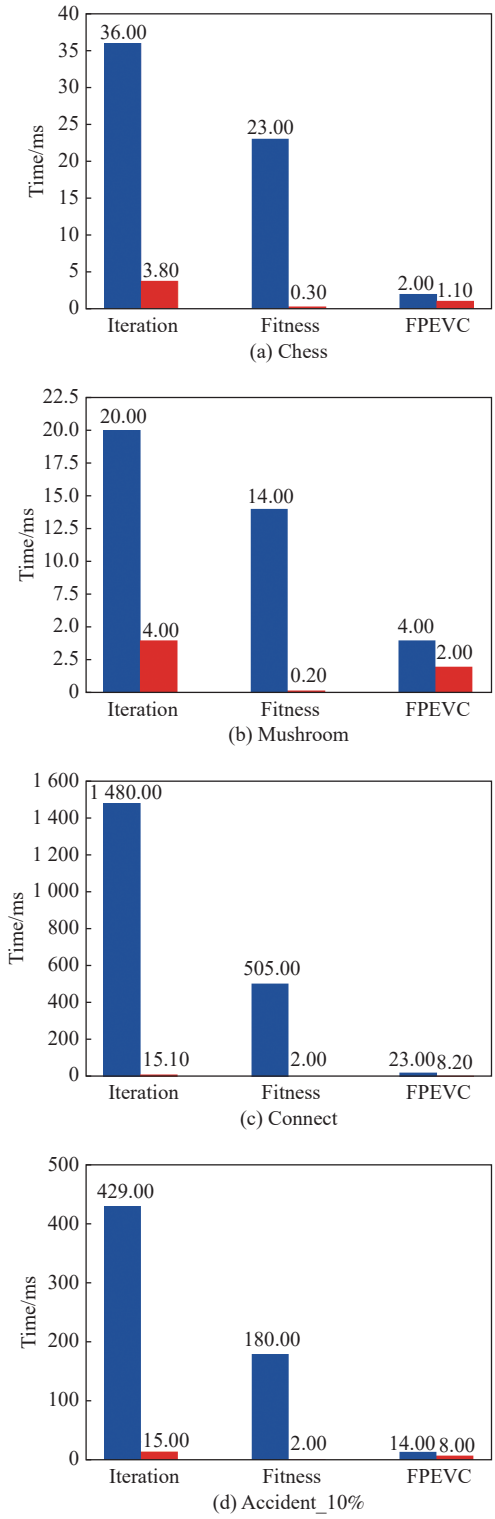


Fig. 7 Comparison on running speed

The results show that fitness evaluation is the most CPU time-intensive operation in each iteration, and the parallel effect on fitness evaluation is obvious. For example, in Mushroom, the running time of fitness evaluation decreases from 14 ms to 0.2 ms, which is 70× acceleration. And in Connect, the accelerate ratio is up to 250× where the running time decreases from 505 ms to 2 ms. The acceleration is benefited from the massive parallel power of GPU by launching a large number of threads per transaction. The FPEVC strategy has serial structure, which raises frequent CPU/GPU synchronization. For example, the acceleration of FPEVC in Chess is only 2×, and the acceleration in Connect is 2.8×. The acceleration of FPEVC on GPU can reduce half the computational time.

Therefore, as shown by the blue and red bars of iteration, the acceleration ratios before and after parallel of the algorithm on four datasets are 9.4×, 5×, 98×, and 28.6×, respectively. The GA steps in CPU have linear time complexity, and no excessive time is added to ensure the mining quality. The experiment shows the effectiveness of GPU acceleration.

5.5 Performance comparison on running speed

In this experiment, the running times of PHUI-GA is compared with HUIM-IGA, HUIF-PSO and PHUI-Miner. The EAs are mining 90% HUIs because they are the approximate algorithms. PHUI-Miner is a CPU parallel exact algorithm and can mine all the HUIs. As shown in Table 4, HUIM-IGA runs faster than HUIF-PSO, that is, the convergence speed of HUIF-PSO is slower and may iterate for more times. Compared with PHUI-GA to HUIM-IGA and HUIF-PSO, the acceleration ratio in datasets with fewer transactions and few items (such as Chess) is 5.59× to 41.2×, and 12× to 102×, respectively. On a dataset with more transactions and more items (such as Connect), the acceleration ratios range from 64× to 76× and 95× to 188×, respectively. Meanwhile, the minimum threshold also affects the acceleration ratio. By varying the minimum threshold from 250 W to 210 W in Accident_10%, the acceleration ratio of PHUI-GA to HUIM-IGA increases in linear speed from 20× to 29×. The acceleration ratio compared to HUIF-PSO is exponential from 21× to 95×. PHUI-Miner runs faster than HUIF-PSO but slower than HUIM-IGA in Chess and Mushroom. With the dataset becomes larger in Accident_10% and Connect, EAs consume a lot of time in fitness calculation and the improved strategies so that the running speed is slower than PHUI-Miner. However, the time cost of PHUI-Miner is long because PHUI-Miner enumerates large itemsets. In all the four datasets, PHUI-GA is up to 30× faster than PHUI-Miner, which shows the advantage of the framework of PHUI-GA with both EAs and GPU parallel.

Table 4 Performance comparison on running speed

Dataset	Minimum utility	PHUI-GA/s	HUIM-IGA/s	Accelerate ratio/%	HUIF-PSO/s	Accelerate ratio/%	PHUI-Miner/s	Accelerate ratio/%
Chess	60 W	0.32	5.59	17.46	12.10	37.81	8.70	27.19
	59 W	0.38	10.40	27.36	18.90	49.73	9.68	25.47
	59 W	0.56	17.80	31.78	28.80	51.42	9.82	17.54
	57 W	0.79	27.56	34.88	57	72.15	10.50	13.29
	56 W	1.07	41.20	38.50	102	95.32	11.10	10.37
Mushroom	46 W	0.52	3.98	7.65	8.20	15.76	6.80	13.08
	45 W	0.70	4.2	6	10.70	15.28	7.10	10.14
	44 W	1.03	4.8	4.66	16.44	15.96	7.20	6.99
	43 W	1.53	7.7	5.03	21	13.72	7.35	4.80
	42 W	2.58	13.6	5.27	28	10.85	7.50	2.91
Accident_10%	250 W	1.47	30	20.41	31	21.08	20.95	14.25
	240 W	2.17	46	21.19	62	28.57	21.74	10.02
	230 W	3.10	72	23.22	214	69.03	23.50	7.58
	220 W	3.75	90	24	300	80	26.60	7.09
	210 W	5.0	149	29.8	477	95.4	27.50	5.50
Connect	1 600 W	3.36	216	64.28	518	154.16	123.00	36.61
	1 590 W	4.10	246	60	572	139.51	135.00	32.93
	1 580 W	5.00	410	82	965	193	152.00	30.40
	1 570 W	6.40	480	75	1 148	179.37	162.00	25.31
	1 560 W	8.31	632	76.05	1 564	188.20	198.00	26.40

6. Conclusions

In this paper, a compute unified device architecture (CUDA) parallel GA is proposed to speed up the EAs on GPU while ensure the mining performance. The improved initialization strategy takes advantage of the item frequency provided by support measure. FPEVC provides more pruning directions to avoid generating many duplicate itemsets. Re-exploration strategy is used to make better use of the discovered HUIs. The GPU parallel on FPEVC and fitness evaluation shows the advantage of GPU acceleration. The experiment shows PHUI-GA reaches the best convergence speed among existing EAs. Through GPU parallel, the accelerate ratio of fitness evaluation step is up to 80×. When mining 90% HUIs, the acceleration of PHUI-GA against the EAs and the CPU parallel implement is up to 188×, which shows both the advantage on improving the mining performance and running speed. Future work may include the study on designing a fully GPU parallel structure to speed up the algorithm with no CPU/GPU interaction.

References

[1] SINGH K, KUMAR R, BISWAS B. High average-utility itemsets mining: a survey. Applied Intelligence, 2022, 52(4): 3901–3938.

[2] JANGRA S, TOSHNIWAL D. Efficient algorithms for victim item selection in privacy-preserving utility mining. Future Generation Computer Systems, 2022, 128: 219–234.

[3] LIN C W, HONG T P, LU W H. An effective tree structure for mining high utility itemsets. Expert Systems with Applications, 2011, 38(6): 7419–7424.

[4] KHEIROLLAHI H, CHAHARDOWLI M, SIMJOO M. A new method of well clustering and association rule mining. Journal of Petroleum Science and Engineering, 2022, 214: 110479.

[5] BICER M, INDICTOR D, YANG R, et al. Efficient implementations for UWEP incremental frequent itemset mining algorithm. International Journal of Applied Logistics, 2021, 11(1): 18–37.

[6] WU P, NIU X Z, FOURNIER-VIGER P, et al. UBP-Miner: an efficient bit based high utility itemset mining algorithm. Knowledge-Based Systems, 2022, 248: 108865.

[7] FOUAD M A, HUSSEIN W, RADY S, et al. An efficient approach for mining reliable high utility patterns. IEEE Access, 2022, 10: 1419–1431.

[8] LEE D, NAM K, HAN I, et al. From free to fee: monetizing digital content through expected utility-based recommender systems. Information & Management, 2022, 59(6): 103681.

[9] SETHI K K, RAMESH D, RATHORE A, et al. HUIM-SMP: high utility itemset mining based stock market analogy. Proc. of the 10th International Conference on Computing, Communication and Networking Technologies, 2019. DOI: 10.1109/ICCCNT45670.2019.8944752.

[10] KRISHNA G J, RAVI V. High utility itemset mining using binary differential evolution: an application to customer segmentation. Expert Systems with Applications, 2021, 181: 115122.

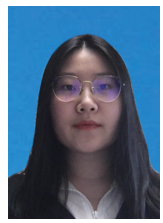
[11] NOUIOUA M, FOURNIER-VIGER P, GAN W S, et al. TKQ: top-K quantitative high utility itemset mining. Proc. of

the Advanced Data Mining and Applications, 2022: 16−28.
[12] LE B, TRUONG T, DUONG H, et al. H-FHAUI: hiding frequent high average utility itemsets. Information Sciences, 2022, 611: 408–431.
[13] HUYNH U, LE B, DINH D T, et al. Multi-core parallel algorithms for hiding high-utility sequential patterns. Knowledge-Based Systems, 2022, 237: 107793.
[14] LIN J C W, GAN W, FOURNIER-VIGER P, et al. High utility-itemset mining and privacy-preserving utility mining. Perspectives in Science, 2016, 7: 74–80.
[15] LIU Y W, WANG L, FENG L, et al. Mining high utility itemsets based on pattern growth without candidate generation. Mathematics, 2021, 9(1): 35.
[16] AHMED C F, TANBEER S K, JEONG B S, et al. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans. on Knowledge and Data Engineering, 2009, 21: 1708−1721.
[17] DUONG Q H, FOURNIER-VIGER P, RAMAMPIARO H, et al. Efficint high utility itemset mining using buffered utility-lists. Applied Intelligence, 2018, 48(7): 1859–1877.
[18] NAWAZ M S, FOURNIER-VIGER P, YUN U, et al. Mining high utility itemsets with hill climbing and simulated annealing. ACM Transactions on Management Information System, 2021, 13(1): 1–22.
[19] CHEN Y, AN A J. Approximate parallel high utility itemset mining. Big Data Research, 2016, 6: 26–42.
[20] SETHI K K, RAMESH D, EDLA D R. P-FHM+: parallel high utility itemset mining algorithm for big data processing. Procedia Computer Science, 2018, 132: 918–927.
[21] LIU J Q, ZHAO R, YANG X C, et al. Efficient parallel algorithm for mining high utility patterns based on spark. Proc. of the IEEE Fourth International Conference on Data Science in Cyberspace, 2019: 367−372.
[22] KANNIMUTHU S, PREMALATHA K. Discovery of high utility itemsets using genetic algorithm with ranked mutation. Applied Artificial Intelligence, 2014, 28(4): 337–359.
[23] LIN J C-W, YANG L, FOURNIER-VIGER P, et al. Mining high-utility itemsets based on particle swarm optimization. Engineering Applications of Artificial Intelligence, 2016, 55: 320–330.
[24] LIN J C W, YANG L, FOURNIER-VIGER P, et al. A binary PSO approach to mine high-utility itemsets. Soft Computing, 2017, 21(17): 5103–5121.
[25] SONG W, HUANG C M. Discovering high utility itemsets based on the artificial bee colony algorithm. Proc. of the Advances in Knowledge Discovery and Data Mining, 2018: 3−14.
[26] HERAGUEMI K E, KAMEL N, DRIAS H. Association rule mining based on bat algorithm. Journal of Computational and Theoretical Nanoscience, 2015, 12(7): 1195–1200.
[27] SONG W, NAN J K. Mining high utility itemsets using ant colony optimization. Proc. of the Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery, 2020: 98−107.
[28] SONG W, LI J Y, HUANG C W. Artificial fish swarm algorithm for mining high utility itemsets. Proc. of the Advances in Swarm Intelligence, 2021: 407−419.
[29] SONG W, HUANG C M. Mining high utility itemsets using bio-inspired algorithms: a diverse optimal value framework. IEEE Access, 2018, 6: 19568–19582.
[30] ZHANG Q, FANG W, SUN J, et al. Improved genetic algorithm for high-utility itemset mining. IEEE Access, 2019, 7: 176799–176813.
[31] ZIDA S, FOURNIER-VIGER P, LIN J C W, et al. EFIM: a highly efficient algorithm for high-utility itemset mining. Proc. of the Advances in Artificial Intelligence and Soft Computing, 2015: 530−546.
[32] SONG W, LI J Y. Discovering high utility itemsets using set-based particle swarm optimization. Proc. of the International Conference on Advanced Data Mining and Applications, 2022: 38−53.
[33] DJENOURI Y, DJENOURI D, BELHADI A, et al. Exploiting GPU and cluster parallelism in single scan frequent itemset mining. Information Sciences, 2019, 496: 363–377.
[34] DJENOURI Y, DJENOURI D, BELHADI A, et al. Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases. Information Sciences, 2019, 496: 326–342.
[35] DJENOURI Y, BENDJOUDI A, MEHI M, et al. GPU-based bees swarm optimization for association rules mining. The Journal of Supercomputing, 2015, 71(4): 1318–1344.
[36] SAFARI M, HUISMAN M. Formal verification of parallel prefix sum and stream compaction algorithms in CUDA. Theoretical Computer Science, 2022, 912: 81–98.

Biographies

JIANG Haipeng was born in 1994. He received his bachelor degree from Jiangnan University, China, in 2020. He is a master student in School of Artificial Intelligence and Computer Science, Jiangnan University. His research interest is high utility itemsets mining based on evolutionary algorithms.
E-mail: 6201924093@stu.jiangnan.edu.cn

WU Guoqing was born in 1984. He received his bachelor degree in automation from China University of Mining and Technology, China, in 2007, master's degree in detection technology and automatic equipment from China University of Mining and Technology, China, in 2010. He then joined China Ship Scientific Research Center, where he has been a senior engineer since 2017. His main research interests include deep-sea equipment test and underwater optical testing technology.
E-mail: www.lotems702@cssrc.com.cn



SUN Mengdan was born in 1995. She received her bachelor degree in mechanical engineering from Shandong University, and master degree in information technology from Monash University in 2021. After graduation, she joined China Ship Scientific Research Center. Her research interests are data analysis and big data processing.
E-mail: sunmengdan702@cssrc.com.cn

LI Feng was born in 1979. He received his bachelor degree in computer Science from Henan University, China, in 2002, and Ph.D. degree from Jiangnan University, China, in 2008. He is now a senior expert in China Ship Scientific Research Center, China. His main research interests include experimental informatization and industrial software.
E-mail: lifeng@cssrc.com.cn

SUN Yunfei was born in 2001. He is an under graduate in Department of Mathematics, Nanjing University. His research interest is data mining based on evolutionary algorithms.
E-mail: 201840049@smail.nju.edu.cn

FANG Wei was born in 1980. He received his Ph.D. degree in Jiangnan University in 2008. He is now a professor of Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Jiangnan University. His research interests are evolutionary algorithm and its applications.
E-mail: fangwei@jiangnan.edu.cn