Hindawi
Mathematical Problems in Engineering
Volume 2023, Article ID 7941673, 15 pages
https://doi.org/10.1155/2023/7941673

Research Article

A High Utility Itemset Mining Algorithm Based on Particle Filter

Yang Yang[1,2] **Jiaman Ding**[3] **Honghai Wang**[1] **Huifen Xing**[1] **and En Li**[1]

[1]*School of Computer and Artificial Intelligence, Chaohu University, Chaohu 238000, Anhui, China*
[2]*Key Laboratory of Data Intelligence and Cyber Security, Chaohu University, Chaohu 238000, Anhui, China*
[3]*Kunming University of Science and Technology, Artificial Intelligence Key Laboratory of Yunnan Province, Kunming 650500, Yunnan, China*

Correspondence should be addressed to Yang Yang; 1023042446@qq.com

Received 1 October 2022; Revised 1 February 2023; Accepted 2 February 2023; Published 23 February 2023

Academic Editor: Breno Jacob

High utility itemset mining is an interesting research in the field of data mining, which can find more valuable information than frequent itemset mining. Several high-utility itemset mining approaches have already been proposed; however, they have high computational costs and low efficiency. To solve this problem, a high-utility itemset mining algorithm based on the particle filter is proposed. This approach first initializes a population, which consists of particle sets. Then, to update the particle sets and their weights, a novel state transition model is suggested. Finally, the approach alleviates the particle degradation problem by resampling. Substantial experiments on the UCI datasets show that the proposed algorithm outperforms the other previous algorithms in terms of efficiency, the number of high-utility itemsets, and convergence.

1. Introduction

Data analysis [1] has recently gained prominence as a field of study in data mining, which has applications in the aviation industry [2], medicine [3], manufacturing [4], and many other fields [5–8]. An essential component of data analysis is frequent itemset mining (FIM) [9], which identifies frequent itemsets (FIs) to provide stakeholders with a basis for decision-making. FIs are a dataset whose occurrence frequency is not less than a user-defined minimum support. However, other valuable datasets with low frequency but high weight cannot be found with conventional FIM. To further filter out high-utility itemsets (HUIs) that can take into account both frequency and weight, high-utility itemset mining (HUIM) [10] has been suggested.

Most HUIM algorithms can be categorized into the following two groups: Apriori-based algorithms and tree structure-based algorithms.

The Apriori [11] is an iterative algorithm that uses the principle of downward closure property to search a dataset multiple times and filter out FIs. In the Apriori algorithm, a dataset is first scanned to obtain candidates. Then, the frequent 1-itemsets are retrieved from these candidates. The new candidates come from the frequent 1-itemsets, and the database is scanned again to find frequent 2-itemsets. All FIs must be determined by scanning the database until no more FIs can be found. Similar to the Apriori method, the Apriori-based HUIM algorithm filters out HUIs by generating candidates at each iteration. For example, Liu et al. [12] suggested a two-stage approach that precisely extracts HUIs by reducing candidates. A construction utility list [13] has also been suggested by Liu and Qu to reduce the generation of candidates.

Since Apriori generates too many candidates and involves multiple scans of a dataset, the FP-growth algorithm is proposed, which requires only two scans of a dataset and is based on a tree structure. In the FP-growth algorithm, a dataset is scanned twice to create an FP-tree, and then conditional FP-trees are created from the FP-tree to extract FIs. The tree structure-based HUIM algorithms, which employ the idea of FP-growth, first store a dataset in a tree structure and then determine HUIs through the conditional tree structure. Recently, many tree structure-based algorithms have been proposed. For example, Yun et al. [14] presented the MU-Growth method, which constructs a MIQ-Tree structure [14] and extracts HUIs using two

effective pruning techniques. Krishnamoorthy [15] presented three pruning strategies to limit the space consumption of the exploration tree for HUIM.

Conventional HUIM algorithms find all HUIs but require complex iterative calculations and take a long runtime. Thus, we propose a particle filter-based high-utility itemset mining (PF-HUIM) algorithm that finds HUIs from a database using a sampling method. The main contributions of this work can be summarized as follows: (1) An efficient algorithm for HUIM based on the particle filter is proposed. To the best of our knowledge, this is the first method to introduce particle filter theory into HUIM. (2) A novel state transition model is proposed, which provides a reasonable sampling distribution. The filtering of HUIs is based on sampling, which significantly reduces the time required by the HUIM algorithm. (3) A particle degradation condition is designed, and then resampling is used to improve the accuracy of the algorithm. (4) Extensive experiments have shown that the particle filter-based HUIM algorithm outperforms three existing algorithms in terms of efficiency, convergence, and the number of HUIs.

The rest of this paper is organized as follows: In Section 2, we give a concise summary of the related work, Section 3 discusses the related theory that is applied in this paper, in Section, we provide a detailed description of the proposed algorithm and examples, Section 5 presents the experiments, and Section 6 gives the conclusion and future work.

2. Related Work

Recently HUIM has become a hot topic, and it is beyond the scope of this paper to thoroughly review the research content of HUIM. Interested readers can consult the relevant literature. In this paper, we give a brief review of the following two different categories of HUIM: Apriori-based HUIM algorithms and tree structure-based HUIM algorithms.

The Apriori-based HUIM approach generates candidates and then filters HUIs from the candidates, which is similar to Apriori. Unlike classical FIM, the downward closure property is no longer applicable to HUIM. Thus, Chan et al. [16] introduced a new pruning strategy for HUIM that allows users to find HUIs from candidates. To ensure that only promising itemsets are added to the candidates, Liu et al. [17] have proposed a transaction-weighted utilization model, which is also applicable to large databases. Yao and Hamilton [18] then designed two innovative pruning strategies and two algorithms for HUIM. These two algorithms improved HUIM performance by reducing the number of candidates. Moreover, an isolated item discarding strategy [19] is presented, which can skip unpromising independent items and further minimize the size of candidates. Such algorithms [12, 13, 16–21] generate too many candidates and require high space complexity.

The tree structure-based HUIM approaches are similar to FP-growth, where the storage of data and the extraction of HUIs are realized through tree structures. For example, CTU-PRO [22] first creates a novel bottom-up utility pattern tree (CUP-tree) and then detects HUIs by traversing this tree. To improve the efficiency of HUIs' discovery, CHUI-Mine [23] has been proposed, which can only capture high-utility candidates from a novel CHUI-Tree [23]. Moreover, Yun and Ryang [24] developed a novel HUPID-tree structure for high-utility pattern mining and designed HUPID-Growth [24] to improve the efficiency of the approach. These trees structure-based HUIM algorithms [14, 15, 22–25] alleviated the problem of excessive time and space consumption in the Apriori-based HUIM algorithms. However, they are expensive to update and create a tree structure. They should be improved.

Unlike the HUIM algorithms described above, this paper designs a particle filter-based HUIM method that uses sampling to filter out HUIs in a dataset. Among the numerous research studies on HUIM algorithms, the heuristic-based HUIM algorithms [26–38] are the most relevant to us. Inspired by the heuristic methods [39, 40], heuristic-based HUIM algorithms first generate random initial candidates, then update the candidates using behavioral patterns of natural organisms, and finally, filter out the HUIs from the candidates.

GA [39] is a classical heuristic method that employs three biologically inspired operators to solve a problem. Kannimuthu and Premalatha [26] proposed two GA-based HUIM methods to search HUIs, which generate initial candidates by random numbers and update the candidates by three operators. Zhang et al. [27] then proposed a novel neighborhood exploration strategy that updates candidates according to the rules of GA.

PSO [40] is another common heuristic method that approximates the solution of the problem by three different operators. Lin et al. [28] presented a PSO-based method called HUIM-BPSOsig that determines current candidates by their previous best position, their global best position, and a sigmoid function. Lin et al. [29] then proposed HUIM-BPSO, which creates an OR/NOR tree based on HUIM-BPSOsig to reduce the computation of unpromising candidates. After that, HUIM-SPSO [30] suggested a bit longer edit distance to find more HUIs and reduce the required runtime. HUIM-IBPSO [31] combines a restart strategy, a modified strategy, a particle movement direction adjustment strategy, and a fitness value hashing strategy with PSO to further improve the performance of HUIM algorithms.

Another heuristic-based algorithm has also been proposed for HUIM. For example, HUIM-HC [32] uses current HUIs as target values for the next population (candidates) to generate HUIs. HUIM-SA [33] then adds an acceptance probability that determines whether the current HUIs are used as target values to avoid the algorithm falling into a locally optimal solution. TKU-CE [34] iteratively finds the top-k HUIs with higher utility values using a probability vector. TKU-CE+ [34] then further improves the accuracy of TKU-CE by ignoring low-utility candidates and randomly generating new itemsets. In addition, HUIM-ABC [35], HUIM-ACS [36], and GWO [37] are all heuristic-based algorithms. They discover HUIs from candidates.

Since heuristic-based algorithms generate candidates based on random numbers or operators, they tend to lose data and lack a rational basis for the candidates generated. Different from the heuristic-based algorithms, we propose a

sampling method based on particle filter theory that uses a novel state transition model to provide a reasonable sampling distribution and resampling to reduce data loss.

3. Related Theory

3.1. High Utility Itemset Mining. Let $I = \{i1, i2, \ldots\ldots, im\}$ be a finite set of m distinct items, and ip denotes an item in I. Each item has an external utility $p(ip)$ that indicates the weight of ip. Let itemset be a set of items all contained in I, and let k-itemset be an itemset containing k items. The x denotes a particle, which in this paper also denotes an itemset. Since the particle filter-based HUIM method needs multiple iterations, we denote the particle generated in the ith iteration (time) as xi, and since multiple particles are generated in each iteration, we denote the jth particle generated in the ith iteration as x_i^j. $DB = \{T1, T2, \ldots\ldots, Tn\}$ denotes a database, where Td denotes a transaction in DB, and the items are contained in I. Each item ip has a corresponding internal utility $q(ip, \mathrm{Td})$ representing the quantity of item ip in transaction Td.

Definition 1. The utility of an itemset x in transaction Td is denoted as $u(x, \mathrm{Td})$ and defined as the following equation:

$$u(x, Td) = \sum_{\mathrm{ip} \in x \wedge x \subseteq Td} u(\mathrm{ip}, Td), \tag{1}$$

where $u(ip, \mathrm{Td}) = p(\mathrm{ip}) \times q(\mathrm{ip}, \mathrm{Td})$. For instance, Table 1 gives a database DB, where T_1 contains items b, c, and e. From Table 1, we can find that $q(b, T_1) = 8$, $q(c, T_1) = 10$, and $q(e, T_1) = 10$. Table 2 shows the external utility of items, therefore, $u(bc, T_1) = p(b) \times q(b, T_1) + p(c) \times q(c, T_1) = 9 \times 8 + 10 \times 1 = 82$.

Definition 2. Utility of an itemset x in a database is denoted as $u(x)$ and is calculated as

$$u(x) = \sum_{x \subseteq Td \wedge Td \in DB} u(x, Td). \tag{2}$$

For instance, $u(bc) = u(bc, T_1) + u(bc, T_3) + u(bc, T_5) + u(bc, T_{10}) = 82 + 62 + 67 + 82 = 293$, $u(ac) = (ac, T_5) + u(ac, T_6) + u(ac, T_8) = 12$.

Definition 3. A high-utility itemset is defined as

$$HUI = \{x \mid u(x) > = \min util\}, \tag{3}$$

where *minutil* represents user-defined minimum support. Let $minutil = 168$, because $u(bc) \geq minutil$, bc is a *HUI*, while $u(ac) < minutil$, and ac is not a HUI.

Definition 4. Utility of a transaction is denoted as $TU(T_d)$ and is calculated as

$$TU(\mathrm{Td}) = \sum_{i \in Td} u(i, Td). \tag{4}$$

For example, $TU(T_1) = u(b, T_1) + u(c, T_1) + u(e, T_1) = p(b) \times q(b, T_1) + (c) \times q(c, T_1) + p(e) \times q(e, T_1) = 162$. Similarly, we can calculate that $TU(T_2) = 68$, $TU(T_3) = 62$, $TU(T_4) = 136$, $TU(T_5) = 68$, $TU(T_6) = 32$, $TU(T_7) = 75$, $TU(T_8) = 67$, $TU(T_9) = 85$, and $TU(T_{10}) = 98$.

Table 1: Database.

T_d	Transactions
T_1	(*b*, 8) (*c*, 10) (*e*, 10)
T_2	(*c*, 8) (*d*, 6) (*f*, 7)
T_3	(*b*, 6) (*c*, 8)
T_4	(*d*, 4) (e, 8) (f, 10)
T_5	(*a*, 1) (*b*, 7) (*c*, 4)
T_6	(*a*, 1) (*c*, 3) (*d*, 4) (*e*, 2)
T_7	(*d*, 5) (*f*, 10)
T_8	(*a*, 1) (*c*, 2) (*e*, 8)
T_9	(*d*, 7) (*e*, 8)
T_{10}	(*b*, 8) (*c*, 10) (*e*, 2)

Table 2: External utility table.

Item	*a*	*b*	*C*	*d*	*e*	*f*
External utility	1	9	1	3	8	6

Definition 5. The transaction-weighted utilization (TWU) of an itemset x is denoted as $TWU(x)$ and is calculated as

$$TWU(x) = \sum_{x \subseteq Td \wedge Td \in DB} \mathrm{TU}(\mathrm{Td}). \tag{5}$$

For instance, $TWU(a) = TU(T_5) + TU(T_6) + TU(T_8) = 167$, Similarly, $TWU(b) = 390$.

Definition 6. High transaction-weighted utilization itemset (HTWUI) is denoted as the following equation:

$$\mathrm{HTWUI} = \{x | TWU(x) > = \min \mathrm{util}\}. \tag{6}$$

Otherwise, x is a low transaction-weighted utilization itemset (*LHTWUI*). Thus, the b is an *HTWUI* since $TWU(b) \geq minutil$. The a is a *LHTWUI* since $TWU(a) < minutil$.

Definition 7. The weight of a particle x is denoted by the following equation:

$$W(x) = \sum_{i=1}^{k} w(i), \tag{7}$$

where k is the number of items contained in a particle. In this paper, the particle x is represented as a bitmap, where each bit represents an item, so a k-itemset is represented by a bitmap of k bits. Let $w(i)$ denote the change probability for bit i of x. Thus, the weight of a particle x is equal to the sum of the change probability for each bit in x. It should be noted that, if we denote a particle by x_i^j, its weight is given by $W(x^j)$.

3.2. Particle Filter. The particle filter is an algorithm that approximates a probability density function by finding a set of random samples that propagate through the state space. It has been widely applied in target tracking [41], indoor positioning [42], indoor navigation [43], urban traffic [44],

battery management systems [45], and other fields. However, there are no studies that have applied the particle filter to HUIM.

To facilitate the subsequent derivation of its prediction and update procedures, the particle filter algorithm first assumes a system. The system consists of a state transition function and an observation function. According to the particle filter theory, the state random variable at time i (x_i) is related to and only related to the state random variable of the previous time, and the observation random variable at time i (y_i) is related to and only related to the state random variable of time i. Let the state transition function f and the observation function h be as follows:

$$\begin{cases} xi = f(xi-1), \\ yi = h(xi). \end{cases} \tag{8}$$

The particle filter algorithm consists of the following main steps: initialization, prediction, updating, and resampling. The purpose of initialization is to generate particles and their weights initially. Prediction and updating predict the prior probability of the next time by updating the particle positions, and the posterior probability is determined by the prior probability and the likelihood probability to correct the particle weights. Resampling is intended to alleviate the problem of particle degradation caused by updating particle weights.

Based on the particle filter algorithm, the proposed algorithm PF-HUIM consists of the following three steps: initialization, state transition, and resampling. PF-HUIM first determines whether particles are HUIs by the likelihood probability of particle sets (population), then designs a state transition model to update the particle positions and correct particle weights, and finally employs resampling to alleviate the particle degradation problem.

3.2.1. Initialization. In PF-HUIM, each iteration (time) generates a population consisting of many particles. Suppose PF-HUIM first randomly generates a population. The population has three particles, ${x_0}^0$, ${x_0}^1$, and ${x_0}^2$, while the corresponding weights of the three particles are $W({x_0}^0)$, $W({x_0}^1)$, and $W({x_0}^2)$, where ${x_0}^0$ represents the first particle at the initial time, and $W({x_0}^0)$ represents the weight corresponding to the first particle at the initial time.

3.2.2. State Transition. Let the prior probability at time i be $p(x_i \mid y_{i-1})$, which is the sampling proposal distribution of particles at time i resulting from the observation at time $i-1$. The observation in PF-HUIM is whether the current particles are HUIs or not. The likelihood probability $p(y_i \mid x_i)$ represents the observation from the sampling proposal distribution of particles at time i in PF-HUIM. The posterior probability $p(x_i \mid y_i)$ represents the sampling proposal distribution of particles at time i due to the observation at time i in PF-HUIM. In the state transition procedure, the prior probability of the next time is first predicted by updating the particles, then the likelihood probability is calculated by the observation of the current time, then the likelihood probability is combined with the prior probability to obtain the posterior probability, and finally, the weights of the particles are corrected by the posterior probability.

Since,

$$\begin{aligned} p(x_i \mid y_{i-1}) &= \int p(x_i, x_{i-1} \mid y_{i-1}) \mathrm{d}x_{i-1} \\ &= \int p(x_i \mid x_{i-1}, y_{i-1}) p(x_{i-1} \mid y_{i-1}) \mathrm{d}x_{i-1} \\ &= \int p(x_i \mid x_{i-1}) p(x_{i-1} \mid y_{i-1}) \mathrm{d}x_{i-1}. \end{aligned} \tag{9}$$

In the above equation, $p(x_i \mid x_{i-1})$ is a state transition function. It is obvious that the prior probability can be derived from the state transition function and the posterior probability.

It is easy to see from the Bayes formula that

$$p(x_i \mid y_i) = \frac{p(y_i \mid x_i) p(x_i \mid y_{i-1})}{\int_{-\infty}^{+\infty} p(y_i \mid x_i) p(x_i \mid y_{i-1}) dx_{i-1}}. \tag{10}$$

Since the infinite integral in the formula given above is difficult to calculate, the posterior probability can be determined by sampling from a set of weighted particles according to particle filter theory. The particles are then obtained from the Monte Carlo sampling distribution and their weights are updated. Now, the posterior probability can be approximately estimated as follows:

$$p(x_i \mid y_i) \approx \frac{1}{N} \sum_{k=1}^{N} \delta(xi - x_i^k) k \in [0, N], \tag{11}$$

where N represents the number of particles sampled at time i, and $\delta(xi - x_i^k)$ is the Dirac delta function. It can be seen that the posterior probability can be approximated by Monte Carlo sampling. So, we need to design a state transition model to give the proposal distribution that can sample and update particles and particle weights.

3.2.3. Resampling. After several particle updates, the variance of particle weights becomes larger and causes the problem of particle degradation. We introduce multinomial resampling to alleviate the problem of particle degradation. The multinomial resampling first calculates the distribution of particle weights, then generates a random number that obeys the uniform distribution, and finally locates the region of the random number in the distribution to obtain resampled particles.

4. A High-Utility Itemset Mining Algorithm Based on Particle Filter

This section describes the proposed PF-HUIM algorithm. First, we briefly describe the overall framework of PF-HUIM. Then, the individual procedures of PF-HUIM are described in detail and illustrated with examples.

4.1. Overall Framework of PF-HUIM. PF-HUIM is proposed based on particle filter theory and finds HUIs from a dataset

by sampling. The overall framework of PF-HUIM, which consists of the following three procedures, is shown in Figure 1.

Initialization: in this procedure, an original database is first filtered, then the filtered database is converted to a bitmap for storage, and finally, an initial population is generated from the bitmap by random sampling. Once the initial population has been generated, it must be determined whether the number of iterations at the current time has reached the maximum number of iterations. If this is the case, the algorithm is terminated. Otherwise, the state transition procedure is followed.

State transition model: this procedure first initializes the change probabilities of the populations, then loops several times to update the populations and their change probabilities, and finally filters the HUIs from the updated populations.

Resampling: after several particle updates, the algorithm suffers from particle degradation. To generate the next population, PF-HUIM resamples the particles from the bitmap with multinomial resampling. Then PF-HUIM goes into the next iteration of state transition and resampling. PF-HUIM outputs all HUIs at the end of the iteration.

4.2. Initialization. Algorithm 1 describes the initialization procedure in detail. In Algorithm 1, *DB* represents a database, *minutil* represents a user-defined minimum support, and *maxiter* represents a maximum number of iterations. Let the database be as shown in Table 1, and let *minutil* = 168. In lines 1–3, each item i in the database with $TWU(i)$ not less than *minutil* is stored in candidates' *SHUIs*. For example, b is stored in *SHUIs* since $TWU(b) \geq minutil$. Similarly, c, d, e, and f are also stored in *SHUIs*. Line 4 removes nonexistent items in *SHUIs* from *DB*. The updated *DB* is then converted into a bitmap in which each item is represented by a bit (Line 5). The bitmap is shown in Table 3. The $IS(DB)$ returns 1 if the item i exists in the updated database, otherwise, it returns 0. Lines 6–7 randomly generate an initial population and let *itertime* = 1. Let the initial population be as shown in Figure 2. It contains three particles $x_0^0 = \{11110\}$, $x_0^1 = \{11010\}$, $x_0^2 = \{01010\}$. Since the top four bits of x_0^0 are 1 and the fifth bit is 0, x_0^0 represents the itemset *bcde*. Similarly, x_0^1 represents *bce* and x_0^2 represents *ce*.

4.3. State Transition Model. After generating an initial population, we propose a state transition model (Algorithm 2) to change the particles in the population and their weights. The model initializes the weight of each particle x in the current population and lets the change probability of each bit in x be 1/2 (lines 1–3). For each particle, the model determines whether its $TWU(x)$ is less than *minutil*. If this is the case, the algorithm calls the procedure hui() (Algorithm 3), otherwise, $u(x)$ determines whether x is *HUIs*. We store x in *HUIs* only if x is a HUI and does not exist in *HUIs* (lines 4–16). The procedure hui() first randomly selects a bit i from x that is 1 to become a 0 bit, then sets the change probability of all 0 bits to 0, and finally decides whether the changed particle should be stored in *HUIs* (lines 2–9). In line 10, the condition for particle degradation is set as $W(x) = 1/2$. If $W(x) = 1/2$, the procedure hui() terminates. An example of a state transition model is shown in Figure 3. Suppose there are three particles in the initial population $x_0^0 = \{11110\}$, $x_0^1 = \{11010\}$, and $x_0^2 = \{01010\}$. Algorithm 2 first sets the change probability of each bit of x_0^0 to 1/2, and then calls the procedure hui() because $TWU(x_0^0) < minutil$. If the first bit in x_0^0 is selected and changed to 0, the updated particle is $x_0^0 = \{01110\}$, then the change probability of the 1st and 5th bits is reduced to 0 and the change probability of the other bit is kept at 1/2. According to equation (2), $u(x_0^0)$ (=31)< *minutil*, x_0^0 is not a *HUI*. The loop then continues since $W(x_0^0) = 3/2$ (according to equation (7)). Since the change probability of 1 bit is 1/2, there is a 1/2 chance that 1 bit will become 0 bit, and 0 bit will never become 1 bit since the change probability of 0 bit is 0. As a result, the particle x will have only one 1 bit ($W(x_0^0) = 1/2$), where x represents 1-*itemsets* and there is no need to find its subset. Therefore, we propose that $W(x) = 1/2$ is the condition for particle degradation. For x_0^1, since $TWU(x_0^1) > minutil$, Algorithm 2 goes to line 7. Since $u(x_0^1)$ (=260) > *minutil* and no x_0^1 is present in *HUIs*, x_0^1 is stored in *HUIs* and the procedure hui() is called.

4.4. Resampling. The relationship between resampling and state transition is shown in Figure 4, and Algorithm 4 represents the entire pseudocode of PF-HUIM. From the state transition model, it is known that the particle degradation problem occurs after the particle is continuously updated. To solve this problem, PF-HUIM must change the particle weights, i.e., reset the change probability of each bit in the particle to 1/2. After that, PF-HUIM resamples the database by multinomial resampling and lets *itertime*++. Finally, the *maxiter* is used to determine whether PF-HUIM should proceed to the next iteration.

5. Experiments

Extensive experiments were conducted to verify the performance of the proposed method PF-HUIM. PF-HUIM was compared with three heuristic-based algorithms which are HUPEumu-GRAM [26], BIO-HUIF-PSO [38], and HUIM-BPSO [30]. HUPEumu-GRAM is a classical GA-based algorithm, HUIM-BPSO is a PSO-based algorithm, and BIO-HUIF-PSO [38] is an optimized PSO-based algorithm. We chose these three algorithms for comparison because they are most similar to PF-HUIM in that they all generate HUIs by selecting candidates. Since the way we proposed to update the population by state transition model is significantly different from the way used by the other three algorithms, the population number in PF-HUIM is not suitable for the analogy with other algorithms. Thus, we do not discuss the population number in this article. In addition, in all of the following experiments, we set the parameters of HUIM-BPSO $w1$ to 0.9, $c1$ to 1, and $c2$ to 2.

PF-HUIM was written in Java and all experiments were conducted on a PC with a 4-core 2.70 GHz CPU, 8 GB RAM,

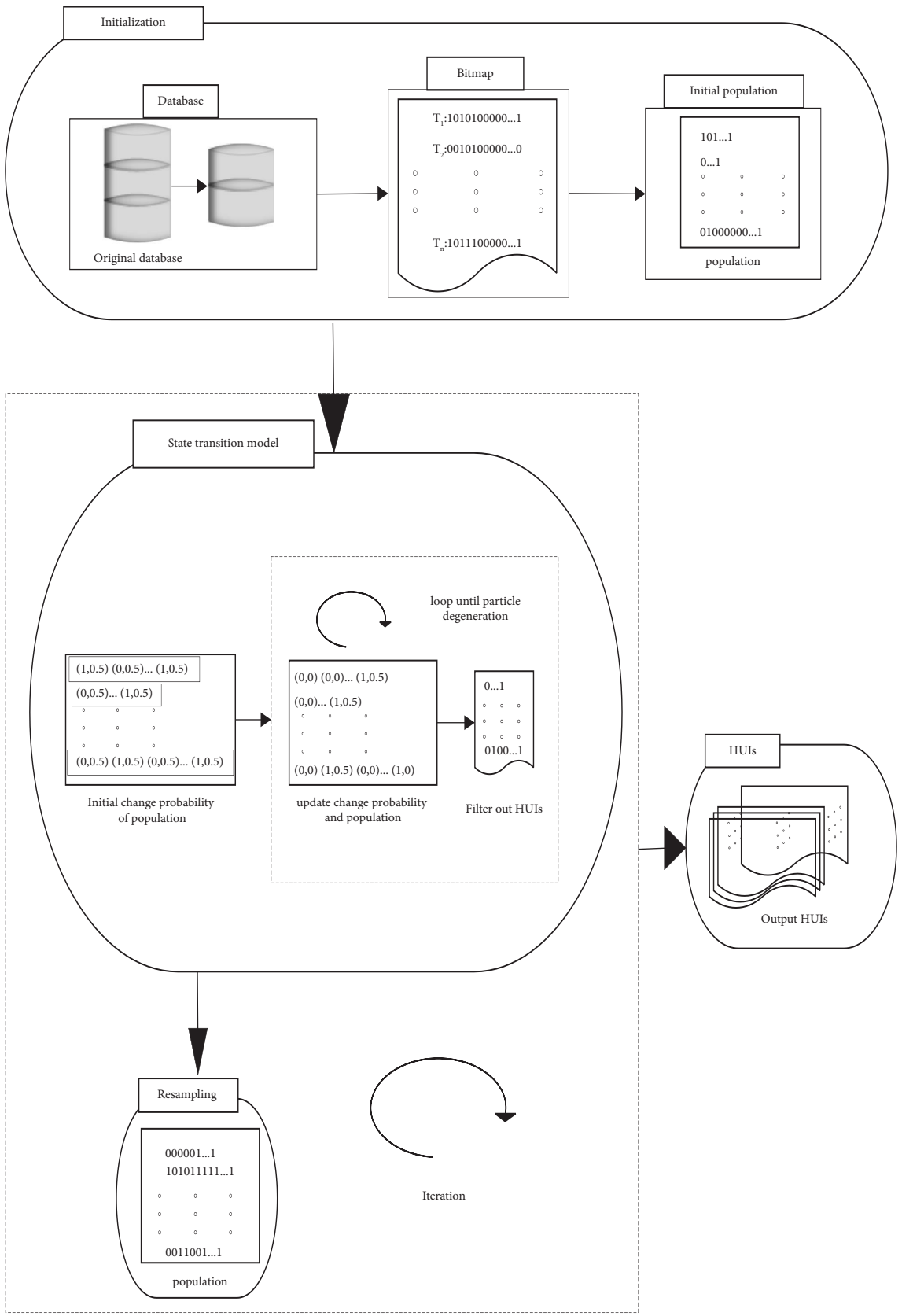

Figure 1: Overall framework of PF-HUIM.

Table 3: Bitmap.

T_d	b	c	d	e	f
T_1	1	1	0	1	0
T_2	0	1	1	0	1
T_3	1	1	0	0	0
T_4	0	0	1	1	1
T_5	1	1	0	0	0
T_6	0	1	1	1	0
T_7	0	0	1	0	1
T_8	0	1	0	1	0
T_9	0	0	1	1	0
T_{10}	1	1	0	1	0

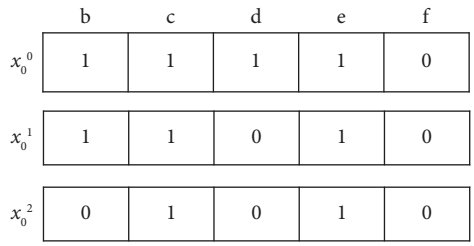

Figure 2: Initial population.

Input: *DB*, *minutil*
Output: population
(1) For each *i DB*
(2) If TWU (*i*) ≥ *minutil*
(3) *i*– > SHUIs
(4) Update (*DB*)
(5) *i* = IS (*DB*)
(6) Initial population
(7) *Itertime* = 1

Algorithm 1: Initializiton.

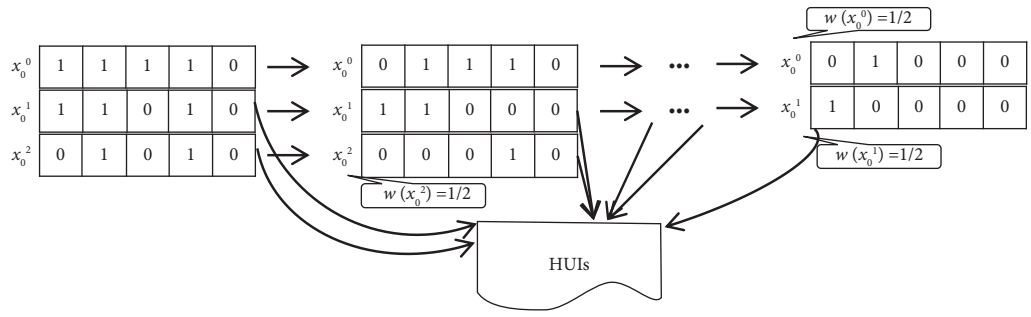

Figure 3: State transition model.

and Microsoft Windows 10 operating system. Table 4 shows the datasets used for these experiments, with each dataset having the following four parameters: transaction count (*T*), average item count per transaction (*A*), and item count (*I*). We note that chess, mushroom, and connect are density datasets, while foodmart, BMS, and crimes in Chicago are sparse datasets. All of these datasets can be downloaded at https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

5.1. Runtime. To validate the efficiency of PF-HUIM, we compare the runtime of HUPEumu-GRAM, BIO-HUIF-PSO, HUIM-BPSO, and PF-HUIM on six datasets.

```
    Input: population
    Output: HUIs
(1) For each x in the population
(2)    For each bit in x
(3)       W (bit) = 1/2
(4) For each x
(5)    If TWU (x) < minutil
(6)          hui()
(7)    else
(8)    {
(9)          if u (x) ≥ minutil
(10)            {
(11)               If x not in HUIs
(12)                  x- > HUIs hui()
(13)            }
(14)         else
(15)            hui()
(16)   }
```

Algorithm 2: State transition model.

```
    Input: x
    Output: HUIs
(1)    Do
(2)       {For each i x
(3)          {if i = = 1
(4)             {Random one i = 0}
(5)          if i = = 0
(6)             {w (i) = 0 }
(7)          if u (x) ≥ minutil
(8)             {if x is not in HUIs
(9)                x- > HUIs}
(10)      }While (w (x) = = 1/2)
```

Algorithm 3: hui().

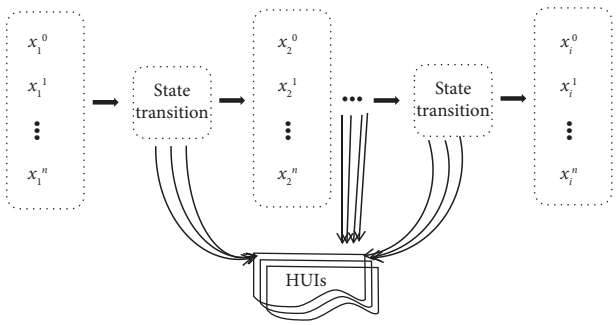

Figure 4: Resampling and state transition.

Figure 5 shows the experimental results of the four algorithms after 1,000 iterations. We can see that the designed algorithm PF-HUIM has the shortest runtime for the same *minutil* compared to the other three algorithms on the dense datasets chess, mushroom, and connect. There are two main reasons for these experimental results. The first reason is that

```
    Input: DB, minutil, maxiter
    Output: HUIs
(1) Algorithm 1
(2) While (itertime ≤ maxiter)
(3)   {
(4)      Algorithm 2
(5)      Multinomial resampling()
(6)      Itertime++
(7)   }
```

Algorithm 4: PF-HUIM.

Table 4: Datasets characteristics.

Dataset name	Transaction count (T)	Average item count per transaction (A)	Item count (I)	Density (%) (A/I) ∗ 100%
Chess	3196	37	75	49.33%
Mushroom	8416	23	119	19.33%
Foodmart	4141	4.42	1559	0.28%
BMS	77512	4.62	3340	0.14%
Connect	67557	43	129	33.33%
Crimes in Chicago	2662309	1.795	35	5.13%

PF-HUIM reduces duplicate calculations by resampling. The second reason is that the state transition model is designed so that the algorithm does not sample the same particles repeatedly. Furthermore, we can see that the runtime of PF-HUIM decreases steadily with increasing *minutils,* while the runtime of the other three methods is not directly related to the *minutil*. This is because the three algorithms use random sampling, whereas the particle filter-based algorithm PF-HUIM uses a more reasonable sampling.

Figures5(c), 5(d), and 5(f) show the experimental results of the four algorithms on the sparse datasets foodmart, BMS, and crimes in Chicago. On BMS and crime s in Chicago, it is obvious that the designed method PF-HUIM still performs the best. However, on the foodmart, BIO-HUIF-PSO is the most efficient and PF-HUIM comes second. The state transition model in FP-HUIM has to repeatedly update the particles (itemsets) that are similar to the original particles. Since the similar itemsets in foodmart are sparse and unevenly distributed, the state transition model requires more runtime. Finally, we note that HUPEumu-GRAM is not shown in Figure 5(c) because its runtime is too long to be compared with the other methods on foodmart.

5.2. Number of HUIs. In this section, the four algorithms are compared on six datasets to verify the number of HUIs generated by PF-HUIM. The number of iterations was set to 1,000 for all experiments, and the results of the experiments are shown in Figure 6.

PF-HUIM can find the most HUIs on three dense datasets. This is because PF-HUIM designs a state transition model based on the particle filter theory. The state transition model updates the particles according to the current particle state. This makes the particle sampling method more

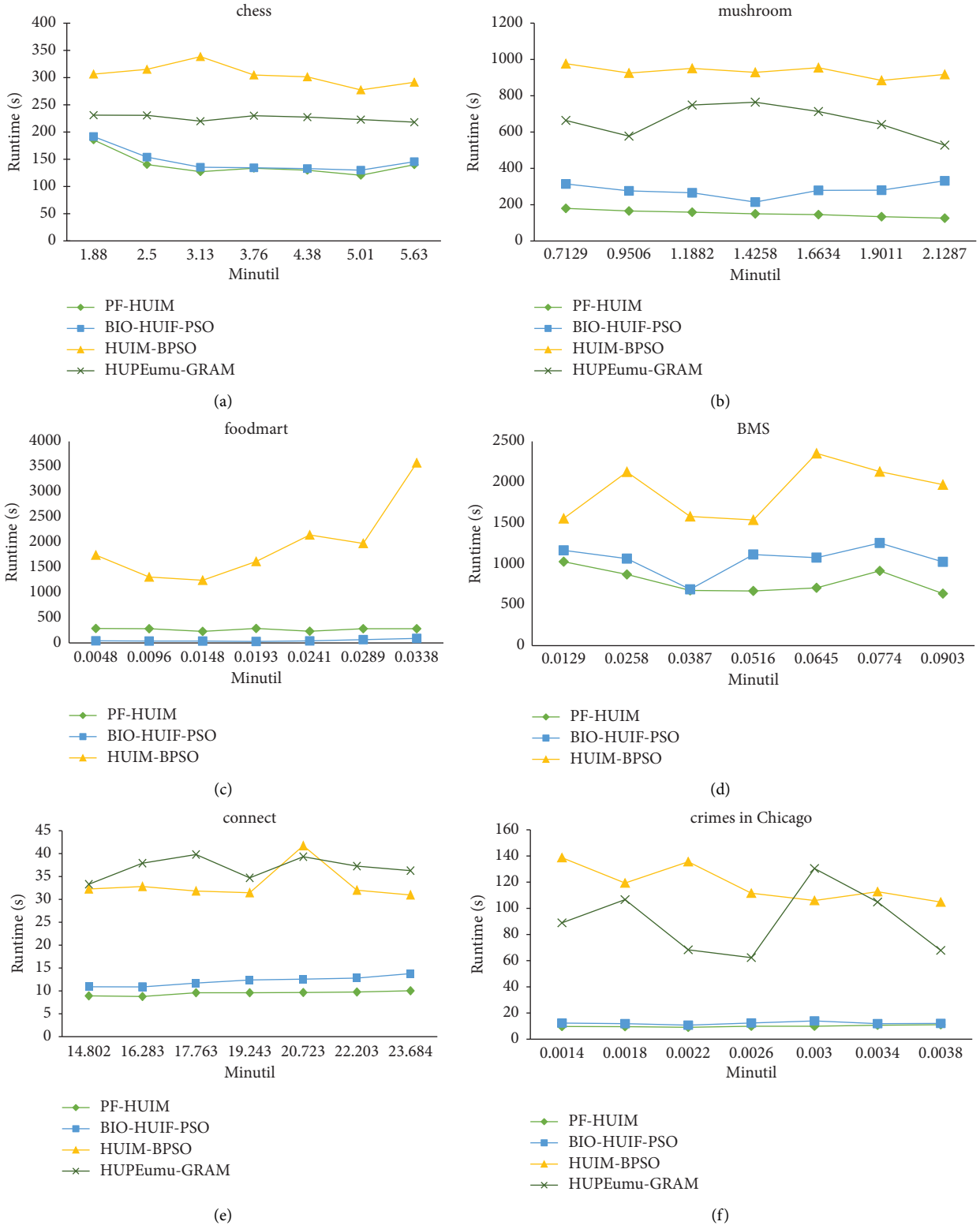

Figure 5: Runtime performance of PF-HUIM against three heuristic-based algorithms.

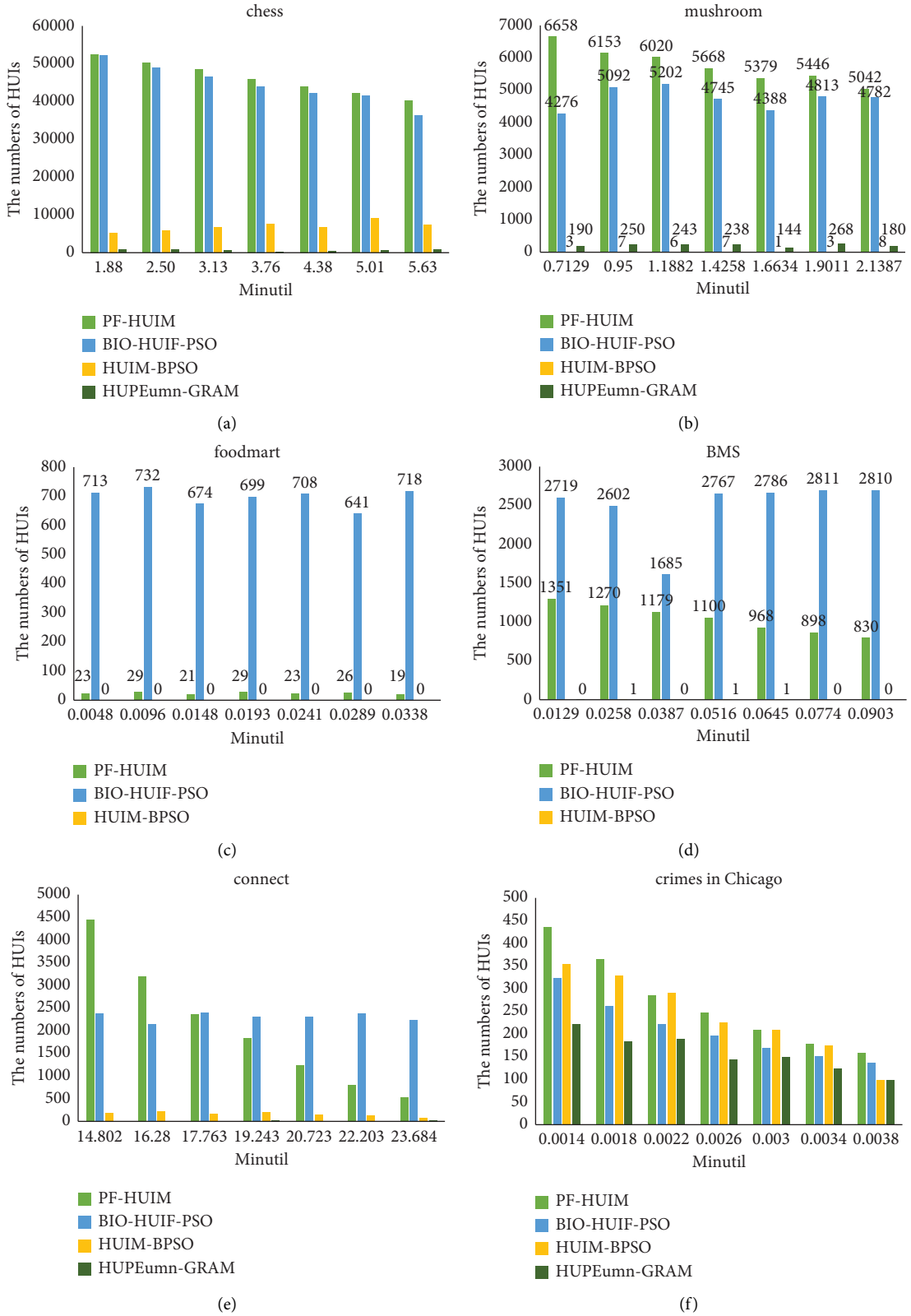

Figure 6: Performance analysis of the numbers of HUIs.

Table 5: The numbers of HUIs.

	Chess	Mushroom	Foodmart	BMS	Connect	Crimes in Chicago
FP-HUIM	52543	6658	23	1351	4445	435
UP-growth	—	10206	37	—	—	650
UP-growth+	—	10206	37	—	—	650
HUI-miner	—	10206	37	—	—	650

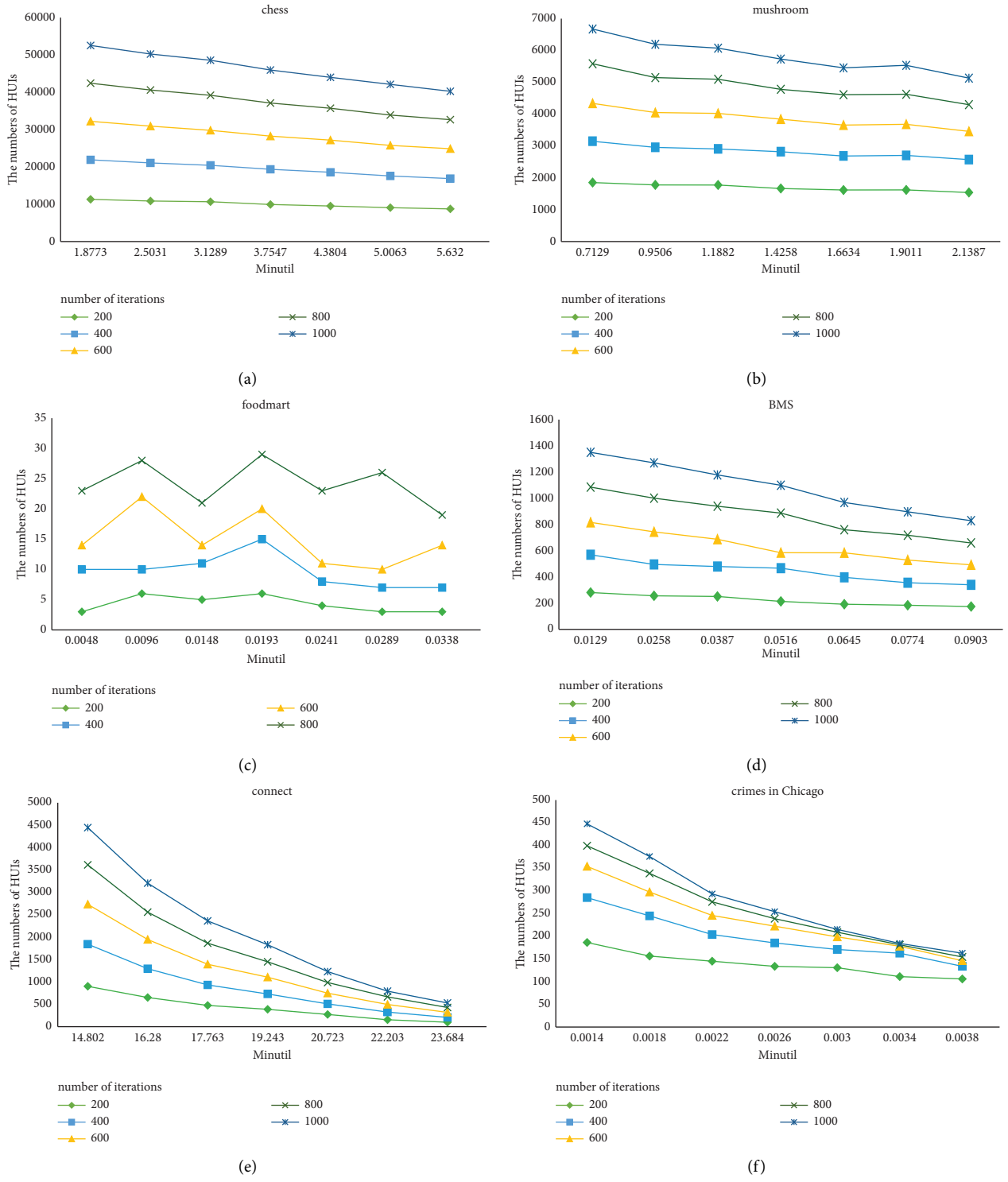

Figure 7: Performance analysis of the iteration numbers.

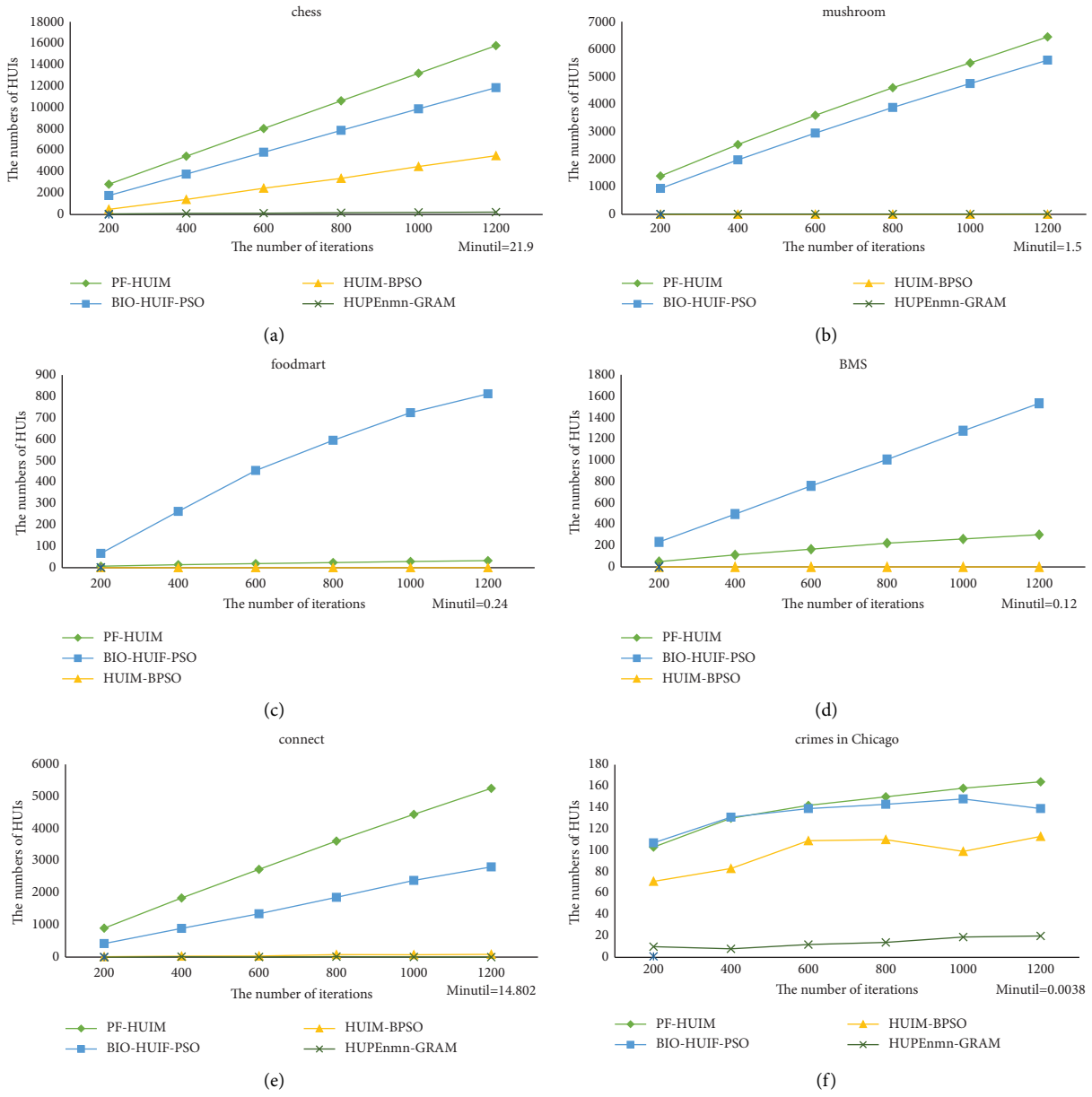

Figure 8: Convergence analysis of PF-HUIM against three heuristic-based algorithms.

reasonable. From the experimental results, we can see that the designed state transition model performs well on dense datasets. In addition, Figure 6 shows that the number of HUIs extracted by PF-HUIM decreases as *minutil* increases, while the relationship between the number of HUIs extracted by the other three algorithms and *minutil* is not significant. In classical HUIM algorithms [12–19, 22–25], the higher the user-defined threshold value, the lower the number of HUIs. These phenomena prove that the proposed method, based on particle filter theory, is rational. The three heuristic-based algorithms, however, rely heavily on random numbers, resulting in algorithms that lack sound bases.

It is not difficult to see that on the sparse datasets from foodmart and BMS, BIO-HUIF-PSO generates the largest number of HUIs, PF-HUIM follows, HUIM-BPSO can barely generate HUIs, and HUPEumu-GRAM takes too much runtime to be shown. There are a large number of long average length transactions in foodmart and BMS, and most similar long average length itemsets are sparsely distributed in the transactions. If the average length of the population is long, the state transition has a tendency to generate many particles that do not exist in the database. This is the reason why PF-HUIM generates fewer HUIs on foodmart and BMS. On the sparse dataset crimes in Chicago, the proposed algorithm PF-HUIM finds the most HUIs when the *minutil* was set to 14.062, 14.802, 14.542, or 17.022, but when the *minutil* is 17.763 or 18.503, PF-HUIM finds slightly fewer HUIs than BIO-HUIF-PSO. These phenomena are due to the fact that the number of HUIs generated from PF-HUIM always decreases as the *minutil* increases. Therefore, we can

see from the experimental results that the designed state transition model is more suitable for dense datasets or sparse datasets with short average lengths.

Since PF-HUIM generated a different number of HUIs than the other three heuristic-based algorithms, we conducted the following experiments. We compared PF-HUIM with three Apriori-based or tree structure-based algorithms which are HUI-Miner [13], UP-Growth [20], and UP-Growth+ [21]. The iteration number of PF-HUIM was set to 1000, the *minutil* for chess was set to 1.88, the *minutil* for mushroom was set to 0.7129, the *minutil* for foodmart was set to 0.0048, the *minutil* for BMS was set to 0.0129, the *minutil* for connect was set to 14.802, and the *minutil* for crimes in Chicago was set to 0.0014. The experimental results are shown in Table 5.

From Table 5 we can see that HUI-Miner, UP-Growth, and UP-Growth + extracted exactly the same number of HUIs on the same datasets. Since HUI-Miner, UP-Growth, and UP-Growth + are 100% accurate, we can assume that FP-HUIM is 64.3% (6658/10206 ∗ 100%) accurate on a mushroom, 60.5% (23/37 ∗ 100%) accurate on foodmart, and 66.92% (435/650 ∗ 100%) accurate on crimes in Chicago. Due to the fact that FP-HUIM generates HUIs by sampling, the accuracy of FP-HUIM is lower than that of the other three algorithms. All experiments were conducted in the same experimental environment. Note that HUI-Miner, UP-Growth, and UP-Growth + give no results on chess, BMS, or connect in Table 5. This is because they are beyond the computational power of this environment. FP-HUIM took 185 seconds to generate 52543 HUIs on chess, 179 seconds to generate 1351 HUIs on BMS, and only 9 seconds to generate 4445 HUIs on connect. Furthermore, FP-HUIM is over 100 times faster than the three algorithms on mushroom, foodmart, and crimes in Chicago. As a result, FP-HUIM is significantly more effective than the other three methods. This is because Apriori-based or tree structure-based algorithms require multiple scans of a dataset and too many calculations.

5.3. Number of Iterations. To verify whether PF-HUIM can find more HUIs by increasing the number of iterations, we set different iterations on six datasets to compare the number of HUIs.

From the experimental results in Figure 7, it can be seen that the number of HUIs produced by PF-HUIM on six datasets increases with the number of iterations. This is because each iteration of PF-HUIM involves a state transition and resampling. The state transition updates the particles, and the resampling generates a new population. The HUIs are filtered from these particles and populations. Therefore, PF-HUIM generates more HUIs by increasing the number of iterations. This means that the accuracy of PF-HUIM can be improved by increasing the number of iterations.

5.4. Convergence. To verify the convergence of the proposed method, we compared four algorithms on six datasets. The *minutil* set by different algorithms for the same dataset is identical to ensure the rigor of the experiment.

From Figures 8(a), 8(b), and 8(e), it can be seen that PF-HUIM performs best on the dense dataset, while HUPEumu-GRAM and HUIM-BPSO can barely detect HUIs. Therefore, the convergence of PF-HUIM has the best performance on dense datasets.

Figures 8(c) and 8(d) show the performance of three algorithms on foodmart and BMS. We do not show the experimental results of HUPEumu-GRAM. This is due to the lack of comparability caused by the long runtime on sparse datasets. It can be observed that PF-HUIM performs second only to BIO-HUIF-PSO. This is because the itemsets in foodmart and BMS are less similar, and PF-HUIM always produces many similar itemsets that do not exist in the database. These itemsets are eventually discarded. On crimes in Chicago, PF-HUIM not only finds more HUIs in the same number of iterations but also ensures that the number of HUIs increases steadily as the number of iterations increases. This is because the average transaction length in crimes in Chicago is short, and PF-HUIM goes to particle degradation rapidly without producing a significant number of duplicate long candidates.

6. Conclusion and Future Work

HUIM provides more valuable information than FIM and has recently become an important study. Apriori-based and tree structure-based algorithms make up the majority of HUIM algorithms. The Apriori-based HUIM algorithms require multiple scans of a dataset to generate candidates. They take a lot of time and memory. The tree structure-based HUIM algorithms are expensive because they need to create and update the tree structures. Thus, we propose an efficient HUIM algorithm based on particle filter theory, which uses a novel state transition model and resampling to solve the problems mentioned above. Numerous experiments have demonstrated that the PF-HUIM discovers HUIs quickly and accurately. However, the performance of PF-HUIM still needs to be improved on sparse datasets with long average lengths. To improve the performance of PF-HUIM on sparse datasets, we will develop more appropriate state transition models and resampling methods in the future.

Data Availability

The .txt type data used to support the experiments in this study and the description of the data can be found at https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the School-Level Scientific Research Project of Chaohu University (Research on frequent pattern mining algorithms for uncertain datasets), (XLY-202108); the Provincial Natural Science Research

Program of Higher Education Institutions of Anhui Province, (KJ2021A1030); the Quality Improvement Project of Chaohu University on Discipline Construction, (KJ21GCZX03); Special Support Plan for Innovation and Entrepreneurship Leaders in Anhui Province; and School-Level Key Scientific Research Projects of Chaohu University (Research on the intelligent elderly service platform around Chaohu Lake based on big data and deep learning), (XLZ-202108).

References

[1] K. Muhammad, M. S. Obaidat, T. Hussain et al., "Fuzzy logic in surveillance big video data analysis: comprehensive review, challenges, and research directions," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–33, 2021.

[2] D. Miltiadou, S. Pitsios, and D. Spyropoulos, "A secure experimentation sandbox for the design and execution of trusted and secure analytics in the aviation domain," *Security and Privacy in New Computing Environments*, vol. 344, pp. 120–134, 2020.

[3] D. Kim, S. Gamboa, and V. Hernandez, "Medical big data analysis system to discover associations between genetic variants and diseases," in *Proceedings of the IEEE International Conference on Communications*, pp. 1–6, IEEE, Montreal, QC, Canada, June 2021.

[4] Y. Lu, Y. Zheng, W. Jing, C. Xu, and Y. Li, "An asynchronous consistency algorithm in smart manufacturing cloud data centers," *Journal of Physics: Conference Series*, vol. 2206, no. 1, Article ID 012020, 2022.

[5] Y. Chen, M. Lin, and D. Zhuang, "Wastewater treatment and emerging contaminants: bibliometric analysis," *Chemosphere*, vol. 297, pp. 133932–133942, 2022.

[6] M. Zhong and M. Lin, "Bibliometric Analysis for Economy in COVID-19 pandemic," *Heliyon*, vol. 8, no. 9, Article ID e10757, 2022.

[7] Y. Luo and M. Lin, "Flash translation layer: a review and bibliometric analysis," *International Journal of Intelligent Computing and Cybernetics*, vol. 14, no. 3, pp. 480–508, 2021.

[8] J. Zhang and M. Lin, "A comprehensive bibliometric analysis of Apache Hadoop from 2008 to 2020," *International Journal of Intelligent Computing and Cybernetics*, vol. 12, 2022.

[9] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347–1362, 2005.

[10] P. Wu, X. Niu, P. Fournier-Viger, C. Huang, and B Wang, "UBP-Miner: an efficient bit based high utility itemset mining algorithm," *Knowledge-Based Systems*, vol. 248, Article ID 108865, 2022.

[11] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487–499, San Francisco, CA, USA, September 1994.

[12] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proceedings of the 1st International Workshop on Utility-Based Data Mining*, pp. 90–99, ACM, Chicago Illinois, August 2005.

[13] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management CIKM*, pp. 55–64, Maui, Hawaii, USA, October 2012.

[14] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3861–3878, 2014.

[15] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[16] R. Chan, Q. Yang, and Y. D. Shen, "Mining High Utility Itemsets," in *Proceedings of the Third IEEE International Conference on Data Mining*, p. 19, IEEE, Melbourne, FL, USA, November 2003.

[17] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proceedings of the 9th pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining PAKDD*, pp. 689–695, Hanoi Vietnam, May 2005.

[18] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, vol. 59, no. 3, pp. 603–626, 2006.

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

[20] V. S. Tseng, C. W. Wu, and B. E. Shie, "UP-growth: An Efficient Algorithm for High Utility Itemset mining," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 253–262, Washington DC USA, July 2010.

[21] V. S. Tseng, B. E. Shie, C. W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772–1786, 2013.

[22] A. Erwin, R. P. Gopalan, and N. R. Achuthan, "A Bottom-Up Projection Based Algorithm for Mining High Utility Itemsets," in *Proceedings of the 2nd International Workshop on Integrating Artificial Intelligence and Data Mining*, pp. 3–11, Gold Coast Australia, December 2007.

[23] W. Song, Y. Liu, and J. Li, "Mining high utility itemsets by dynamically pruning the tree structure," *Applied Intelligence*, vol. 40, no. 1, pp. 29–43, 2014.

[24] U. Yun and H. Ryang, "Incremental high utility pattern mining with static and dynamic databases," *Applied Intelligence*, vol. 42, no. 2, pp. 323–352, 2015.

[25] D. Kim and U. Yun, "Efficient algorithm for mining high average-utility itemsets in incremental transaction databases," *Applied Intelligence*, vol. 47, no. 1, pp. 114–131, 2017.

[26] S. Kannimuthu and K. Premalatha, "Discovery of high utility itemsets using genetic algorithm with ranked mutation," *Applied Artificial Intelligence*, vol. 28, no. 4, pp. 337–359, 2014.

[27] Q. Zhang, W. Fang, J. Sun, and Q. Wang, "Improved genetic algorithm for high-utility itemset mining," *IEEE Access*, vol. 7, pp. 176799–176813, 2019.

[28] J. C. W. Lin, L. Yang, P. Fournier-Viger et al., "Mining high-utility itemsets based on particle swarm optimization," *Engineering Applications of Artificial Intelligence*, vol. 55, pp. 320–330, 2016.

[29] J. Lin, L. Yang, P. Fournier-Viger, T. P. Hong, and M. Voznak, "A binary PSO approach to mine high-utility itemsets," *Soft Computing*, vol. 21, no. 17, pp. 5103–5121, 2017.

[30] W. Song and J. Li, "Discovering high utility itemsets using set-based particle swarm optimization," *ADMALNCS*, vol. 12447, pp. 38–53, 2020.

[31] W. Fang, Q. Zhang, H. Lu, and J. Lin, "High-utility itemsets mining based on binary particle swarm optimization with

multiple adjustment strategies," *Applied Soft Computing*, vol. 124, Article ID 109073, 2022.

[32] S. Bagui and P. Stanley, "Mining frequent itemsets from streaming transaction data using genetic algorithms[J]," *Journal of Big Data*, vol. 7, no. 1, pp. 1–20, 2020.

[33] M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, and W. Song, "Mining high utility itemsets with hill climbing and simulated annealing," *ACM Transactions on Management Information Systems*, vol. 13, no. 1, pp. 1–22, 2021.

[34] W. Song, C. Zheng, and C. Huang, "Heuristically Mining the Top-K High-Utility Itemsets with Cross-Entropy optimization," *Applied Intelligence*, vol. 52, pp. 1–16, 2021.

[35] W. Song and C. Huang, "Discovering high utility itemsets based on the artificial bee colony algorithm," *Pacific-Asia Conference on Knowledge Discovery & Data Mining*, vol. 10939, pp. 3–14, 2018.

[36] J. M. T. Wu, J. Zhan, and J. C. W. Lin, "An ACO-based approach to mine high-utility itemsets," *Knowledge-Based Systems*, vol. 116, no. 15, pp. 102–113, 2017.

[37] N. Pazhaniraja, S. Sountharrajan, and B. Sathis Kumar, "High utility itemset mining: a Boolean operators-based modified grey wolf optimization algorithm," *Soft Computing*, vol. 24, no. 21, pp. 16691–16704, 2020.

[38] W. Song and C. Huang, "Mining high utility itemsets using bio-inspired algorithms: a diverse optimal value framework," *IEEE Access*, vol. 6, no. 1, pp. 19568–19582, 2018.

[39] D. E. Golberg, *Genetic Algorithms in Search Optimization and Machine learning*, Addion wesley, no. 102, Boston, MA, USA, 1989.

[40] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[41] X. Lu, F. Li, and J. Tang, "A new performance index for measuring the effect of single target tracking with Kalman particle filter[J]," *International Journal of Modern Physics C*, vol. 33, no. 09, pp. 1–14, 2022.

[42] W. Shao, F. Zhao, H. Luo, H. Tian, J. Li, and A. Crivello, "Particle filter reinforcement via context-sensing for smartphone-based pedestrian dead reckoning," *IEEE Communications Letters*, vol. 25, no. 9, pp. 3144–3148, 2021.

[43] G. Raja, S. Suresh, S. Anbalagan, A. Ganapathisubramaniyan, and N. Kumar, "PFIN: an efficient particle filter-based indoor navigation framework for UAVs," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4984–4992, 2021.

[44] L. Wei, Y. Wang, and P. Chen, "A particle filter-based approach for vehicle trajectory reconstruction using sparse probe data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, pp. 2878–2890, 2021.

[45] Y. Bi, Y. Yin, and S. Y. Choe, "Online state of health and aging parameter estimation using a physics-based life model with a particle filter," *Journal of Power Sources*, vol. 476, Article ID 228655, 2020.