IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 May 2024, accepted 9 June 2024, date of publication 13 June 2024, date of current version 21 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3413966

RESEARCH ARTICLE

Frequent Closed High-Utility Itemset Mining Algorithm Based on Leiden Community Detection and Compact Genetic Algorithm

XIUMEI ZHAO, XINCHENG ZHONG, AND BING HAN
Department of Computer, Changzhi University, Changzhi 046000, China

Corresponding author: Xiumei Zhao (cy11997093@czc.edu.cn)

This work was supported in part by the Research Project of Changzhi University in 2023 under Grant XN0717; in part by the Fundamental Research Program of Shanxi Province under Grant 202103021223381.

ABSTRACT Traditional pattern mining algorithms are based on tree and linked list structures. However, they often only consider a single factor of frequency or utility and have to deal with exponential search spaces as well as generate numerous candidates. Thus, we propose a frequent closed high-utility itemset mining (FCHUIM) algorithm based on Leiden community detection and a compact genetic algorithm (LcGA). This algorithm first employs Leiden community detection to decompose a dataset into several highly related transaction communities and then uses an approximate or exact strategy to mine frequent itemsets. Subsequently, it checks for closures in the mined frequent itemsets and finally employs the compact genetic algorithm to efficiently mine high-utility patterns from the frequent closed itemsets. Experimental results on four real datasets, namely Retail, Chainstore, Chess, Accidents,among which the datatype of the former two is Sparse and that of the latter two is Dense, demonstrate that compared with modified traditional closed high-utility mining algorithms, including CHUI-Miner-S, CLS-Miner-S and the most advanced algorithms for mining frequent closed high-utility mining algorithms called FCHUIM, the average runtime of LcGA-FCHUIM is lower by 40.5% than the optimal contrastive algorithm. Moreover, LcGA-FCHUIM can mine an average of 96% of frequent closed high-utility itemsets, making it an effective algorithm for frequent closed high-utility itemset mining and suitable for most scenarios.

INDEX TERMS Frequent closed high-utility itemset mining (FCHUIM), Leiden community detection, compact genetic algorithm (cGA), approximate strategy, exact strategy.

I. INTRODUCTION

The objective of data mining is to extract effective, unusual, potentially valuable, and interpretable patterns from massive data. The primary tasks include predictive and descriptive processing, offering specific sales strategies or other decision-making information. Frequent itemset mining (FIM) [1] is a descriptive task applied in market basket analysis to extract frequently occurring patterns for promotional purposes. However, FIM is limited by two assumptions. First, it assigns equal importance to all items without considering other semantic factors, such as profit and cost. Second, it only considers whether an item exists in a transaction and does not reflect the original purchase quantity in that transaction. For instance, the pattern {instant noodles, cola} may be frequent, but it may yield very limited profit because supermarkets often use cola as a low-price promotional item. Conversely, the pattern {diamond ring, Maotai liquor} may not be frequent, but it can generate substantial profit. To address these limitations, high-utility itemset mining (HUIM) was introduced [2]. The utility of items can be evaluated based on different criteria, such as the number of website visits, the profit from selling items, and other user-specified standards. High-utility itemsets (HUIs)

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Saleem.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

are itemsets with utility function values that are not less than the user-specified minimum utility threshold. Mining based on utility patterns has widespread applications, such as website clickstream analysis, mobile e-commerce, biomedical applications, and cross-marketing [3].

HUIM algorithms based on tree structures [4] and evolutionary models [5] often produce a large number of HUIs, which can sometimes be challenging for users to comprehend. Further, numerous HUIs can deteriorate an algorithm's execution time and memory usage [6]. Thus, efforts have been made to generate compact and lossless HUI representations, known as closed high-utility itemsets (CHUIs) [7]. An itemset is a CHUI if it satisfies two conditions: 1) it is an HUI and 2) it has no supersets with the same support. Therefore, CHUI mining (CHUIM) can provide comprehensive yet more concise information for decision-makers. The first CHUIM algorithm was proposed in [8]. This algorithm uses a two-phase approach to mine information. However, the first phase generates numerous candidate sets and the second phase requires repeated dataset scanning. Thus, its mining efficiency needs improvement. Reference [9] used an extended utility list to maintain itemsets discovered in transactions and applied a divide-and-conquer model to mine complete CHUIs. However, operations to maintain the utility list are time-consuming, and its pruning strategy uses a loose upper bound. Thus, overall efficiency still needs improvement [10], after analyzing the strengths and weaknesses of reference [9], proposed strategies such as link-estimated utility co-occurrence pruning and closure detection. The modified algorithm based on the combined effect of these strategies is a state-of-the-art CHUIM algorithm. However, it cannot effectively handle large-scale datasets [11].

Traditional CHUIM algorithms typically consider finding all required CHUIs. First, such algorithms are unsuitable for making online recommendations in a short time, such as in the stock market. Second, mined itemsets need further analysis to be effectively used, and the interpretation of CHUIs may require domain knowledge; otherwise, they might not be intuitive. Finally, some CHUIs may not be frequent enough to be of significant reference value for most merchants. In real life, only a few merchants focus on the sales of luxury goods. Best-selling products are more favored by merchants because they can quickly recirculate funds to reduce unnecessary financial risk issues. Therefore, people are more interested in mining frequent CHUIs (FCHUIs) [12], [13] because mining a vast majority of FCHUIs in a short time can also meet daily decision-making needs, and its mining results are more intuitive, i.e., it has stronger interpretability. In other words, we need to mine an itemset that can simultaneously meet or exceed the minsup and minutil thresholds in a transactional dataset and has a concise representation. Here, minsup and minutil are specified by users. An efficient algorithm for mining frequent closed high-utility itemsets (FCHUIM) was proposed in [14]. This algorithm employs strategies such as extended utility pruning upper bounds, a total counting list structure, and prechecking to accelerate the mining of target itemsets. Although it is efficient to a certain extent, it remains a mining algorithm based on a linked list structure. A solution to address the above limitations is to first consider dataset decomposition, then find all frequent closed itemsets, and finally use a fast search algorithm to find HUIs among the frequent closed itemsets.

At present, there is no research on the combination of the Leiden community detection algorithm and compact genetic algorithm (cGA) for FCHUI mining (FCHUIM). To improve mining efficiency, we propose a CHUIM algorithm based on the Leiden community detection algorithm and compact genetic algorithm, with the following key features: 1) it uses the Leiden community detection algorithm to decompose a dataset and minimizes interclass shared items, thereby obtaining knowledge discovered as a training set for subsequent prediction or mining with higher accuracy; 2) it uses cGA to efficiently mine FCHUIs in a short time, benefiting online decision-making for large-scale databases. Extensive experiments on various large-scale datasets demonstrate that the proposed algorithm outperforms modified CHUI-Miner and chain lower branch strategy miner (CLS-Miner) algorithms.

II. RELATED WORK

A. COMMUNITY DETECTION

Community detection is commonly used to understand complex network structures. The Louvain algorithm [15] is a widespread model for revealing community structures. However, it has some limitations such as low accuracy in community partitioning, sensitivity to the distribution of cell sizes within groups, and the identification of small subgroups with no connections within the same cluster. To address these limitations, the Leiden community detection algorithm was proposed in [16]. This algorithm is a hierarchical clustering approach that optimizes modularity through greedy moves, recursively merging communities into single nodes and repeating the process on a compressed graph. The algorithm involves three stages: local movements of nodes, refinement of partitions, and aggregation of networks based on the refined partitions. The Leiden community detection algorithm ensures strong connections between communities, has higher speed and better scalability than the Louvain algorithm, and can operate on graphs with millions of nodes [17].

Transaction databases also possess community properties, with each database representing a collection of similar transactions. Reference [18] used various algorithms to decompose transaction databases. However, the algorithms' results were affected by the decomposition quality and several shared items were observed between clusters. If there is no shared item between clusters, the number of patterns discovered with and without decomposition is the same [19].

B. cGA

Several evolutionary algorithms have been proposed for mining association patterns [20], [21]. However, they have limitations in terms of runtime and accuracy, especially for large datasets. To address these limitations, we introduce cGA to explore transaction communities. Although cGA is a so-called genetic algorithm, it is essentially an estimation of distribution algorithm (EDA) [22]. It has similar accuracy to that of genetic algorithms but requires less time and memory, making it more efficient. Compared with other EDAs, cGA has a more concise probability model and a smaller population size. In cGA, a candidate solution is represented by a probability vector, and two individuals are generated based on this probability vector to compete. The probability vector is then updated according to a specific strategy. The cGA process is simple and does not require large-scale crossover and mutation operations as in traditional genetic algorithms, significantly reducing storage space and improving efficiency.

The cGA workflow is as follows (**Steps 1-6**):

Step 1: Initialize the probability vector, typically setting the probability for each gene to 0.5.

Step 2: Sample using the probability model to generate two individuals.

Step 3: Evaluate the fitness of the two individuals.

Step 4: Select the individual with a higher fitness as the winning individual.

Step 5: Update the probability model using the winning individual.

Step 6: Repeat Steps **2-5** until the termination condition is met.

C. CHUIM

Existing studies mostly focus on FIM and HUIM, with research on FCHUIM being relatively rare. The basic context of utility pattern mining has been described above. In this section, we only provide a review of the research progress on CHUIM, which is most relevant to this study. Because HUIM generates a large number of results, which are time-consuming for users to analyze, a more concise itemset representation, that is, CHUIM, is necessitated. The concept was first proposed by Tseng et al. [8], who incorporated four effective strategies to reduce the number of candidate itemsets. However, their algorithm's two-phase mining process requires additional database scans and is quite time-consuming in practice. Thus, Wu et al. [19] proposed CHUI-Miner, a one-phase mining algorithm that adopts a special structure called EU-List to store the utility information of itemsets without additional database scans, effectively improving mining efficiency. However, CHUI-Miner only uses TWU and the remaining utility as pruning strategies and several candidate itemsets are still retained in the search process. Another CHUIM algorithm, named EFIM-Closed, was proposed in [7]. Two tighter upper bounds and a method for forward and backward closure checking were introduced to effectively prune more search spaces, but there is a drawback of needing to repeatedly scan a database when the dataset is dense. CLS-Miner was proposed in [10]. CLS-Miner is similar to CHUI-Miner but employs more effective strategies, such as Chain-EUPC, coverage, LBP, and an improved subsume relationship check method, making it a state-of-the-art CHUIM algorithm. Reference [23] used a multi-objective model to mine CHUIs, initially clustering with the k-means model and then using the MapReduce model and cGA to test potential and probable candidates for mining large-scale CHUIs in a large database, significantly improving mining efficiency. However, the model is not a complete algorithm and its mining accuracy needs improvement. FCHUIs, as a compact form of expression, are not only few in number but can also provide lossless information. Reference [14] studied FCHUIs using a method similar to CLS-Miner by introducing an extended utility pruning upper bound strategy, which is tighter than the traditional pruning upper bounds and can effectively mine the research targets. Nevertheless, it is still time-consuming. In this study, we aim to achieve a balance between mining efficiency and accuracy.

III. PREPARATION OF RELEVANT KNOWLEDGE

In this section, we provide a detailed introduction to the relevant knowledge and definitions involved in this study.

Let $I = \{i_1, i_2, \ldots i_k\}$ be a finite set of items and the transaction database $D = \{T_1, T_2, \ldots T_n\}$ comprises transaction information and external utility tables. The external utility table stores the unit profit or importance of each item, and the transaction information table records the frequency of occurrence of each item in each transaction and the utility value of each transaction. Each transaction is a subset of item collection I, *i.e.*, $T_c \subseteq I$, and each transaction T_c has a unique identifier T_{id}. Table 1 presents a small quantitative transactional database, and Table 2 lists the profit of different items in this database.

TABLE 1. Transaction information.

Number	Business	Transactional utility
T_1	a:1, c:18, e:1	27
T_2	d:1, e:1	11
T_3	c:4, e:2	16
T_4	a:1, b:1, f:3	15
T_5	a:3, c:25, d:3, e:1	55
T_6	b:1, f:2	11

TABLE 2. External utility.

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

Definition 1 (Frequent Itemset (FI)): For any given itemset X, if its support is greater than or equal to a given support

threshold min*Sup*, *i.e.*, $\sup(X) \geq$ minSup, then X is called an FI.

Definition 2 (Utility of a Single Item in a Transaction): The utility of item i_j in transaction data T_q, denoted by $u\left(i_j, T_q\right)$, is defined

$$u\left(i_j, T_q\right) = q\left(i_j, T_q\right) \times p\left(i_j\right). \tag{1}$$

For example, in Table 1,

$$u(c, T_3) = q(c, T_3) \times p(c) = 4 \times 1 = 4,$$
$$u(e, T_5) = q(e, T_5) \times p(e) = 1 \times 6 = 6.$$

Similarly, the utility of itemset X in transaction T_q is defined as follows:

$$u(X, T_q) = \sum_{i_j \in T_q \wedge X \subseteq T_q} u(i_j, T_q). \tag{2}$$

For example,

$$u(\{a, c\}, T_5) = q(a, T_5) \times p(a) + q(c, T_5) \times p(c)$$
$$= 3 \times 3 + 25 \times 1 = 34.$$

Definition 3: (Utility of a Single Itemset in the Database): The utility of itemset X in database D, denoted by $u(X)$, is used as the fitness function for the subsequent cGA HUIM. It is defined as follows:

$$u(X) = \sum_{i_j \in T_q \wedge X \subseteq T_q} u(X, T_q). \tag{3}$$

For example,

$$u(\{a, c\}) = u(\{a, c\}, T_1) + u(\{a, c\}, T_5)$$
$$= u(a, T_1) + u(c, T_1) + u(a, T_5) + u(c, T_5)$$
$$= 3 + 18 + 9 + 25 = 55.$$

Definition 4: (Utility of a Single Transaction in the Database): The utility of transaction T_q in database D, denoted by $tu(T_q)$, defined as follows:

$$tu(T_q) = \sum_{i_j \in T_q} u(i_j, T_q). \tag{4}$$

For example,

$$tu(T_6) = u(b, T_6) + u(f, T_6) = 1 \times 9 + 2 \times 1 = 11.$$

Definition 5 (Total Utility of the Database): The total utility of transaction database D is the sum of the utilities of individual transactions:

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

For example,

$$U(D) = tu(T_1) + tu(T_2) + tu(T_3)$$
$$+ tu(T_4) + tu(T_5) + tu(T_6)$$
$$= 27 + 11 + 16 + 11 + 55 + 15 = 135.$$

Definition 6: (HUIM): Assuming the minimum utility threshold coefficient is δ, itemset X is considered an HUI if its utility value is not less than the minimum utility value, min*Uti*:

$$u(X) \geq \min Uti = \sum_{T_q \in D} tu(T_q) \times \delta \tag{6}$$

For example, if $\delta = 30\%$, then min$Uti = 135 \times 30\% = 40.5$. Because $u(\{a, c\}) = 55 > 40.5$, $\{a, c\}$ is an HUI; meanwhile, because $u(\{a, b, f\}) = 15 < 40.5$, a, b, f is not an HUI.

Definition 7: (CHUI): Given itemset X. It is considered a CHUI if and only if it has no proper superset Z such that $\sup(X) = \sup(Z)$ (in other words, X and Z are HUIs, *i.e.*, $u(X) \geq \min Uti$, $u(Z) \geq \min Uti$).

Definition 8: (FCHUI): Given itemset X. It is considered an FCHUI if and only if its support value $\sup(X) \geq minSup$, its utility value $u(X) \geq \min Uti$, and it is closed.

For example, if min$Sup = 33.3\%$ and min$Uti = 30$, then $\{d, e\}$ is an FCHUI because it has a support of exactly 33.3%, $u(\{d, e\}) = 32 > 30$, and there are no supersets of $\{d, e\}$ with the same support. Meanwhile, $\{c, e\}$ is not an FCHUI. Although it satisfies the minimum support and minimum utility requirements, its superset $\{a$, c, e$\}$ has the same support and satisfies the first two minimum requirements. The purpose of FCHUIM is to discover combinations of items in transaction databases that are both frequent and have high utility while still being concise in their representation. These patterns can assist market decision-makers in developing rational and effective sales strategies.

IV. FCHUI BASED ON cGA

In this section, we introduce the proposed Leiden community detection and cGA-based FCHUIM algorithm (LcGA-FCHUIM). The algorithm begins by using the Leiden community detection method to decompose the transaction database into highly related transaction communities, thereby reducing the search space. It then proceeds to mine frequent closed itemsets within each community and finally uses cGA to mine HUIs within the frequent closed itemsets.

A. LEIDEN COMMUNITY DETECTION

First, the transaction database is transformed into a graph network, where each transaction is considered a node, and if two transactions share items, an edge is added between the corresponding nodes. The graph $G =< V, E >$ is constructed from this transaction database D and itemset I, where V and E denote the vertex and edge sets, respectively. The transformation process is given by (7) and (8):

$$\forall T_i \in D, \forall V_i \in V, V_i = T_i \tag{7}$$

$$\forall (T_i, T_j), \exists i_k \in I, i_k \in T_i \vee i_k \in T_j, E_{ij} = 1$$
$$\forall (T_i, T_j), \exists i_k \in I, i_k \in T_i \vee i_k \in T_j, E_{ij} = 1. \tag{8}$$

For example, in the transaction database shown in Table 1, if we consider the first four transactions, V is constructed from four transactions, *i.e.*, $V = D = \{V_1, V_2, V_3, V_4\}$. If two vertices share items, an edge is created between them.

E is defined as $E = \{E_1, E_2, E_3, E_4\}$, where $E_1 = \{V_1, V_2\}$, $E_2 = \{V_1, V_3\}$, $E_3 = \{V_1, V_4\}$, and $E_4 = \{V_2, V_3\}$. The graph network G corresponding to the transaction database in Table 1 is shown in Fig. 1.

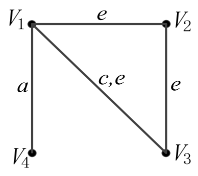

FIGURE 1. The graph network corresponds to Table 1.

After creating the graph network corresponding to the transaction database, the Leiden community detection algorithm can be used to construct highly related transaction communities by separating nodes into disconnected clusters. Thus, the modularity score of each community can be maximized, where modularity quantifies the quality of node assignment to communities. In each iteration, the algorithm optimizes modularity through greedy moves, recursively merging communities into single nodes and repeating this process on a compressed graph. The result is a set of communities, each of which is a highly related subgraph. In this context, a community corresponds to a set of highly related transactions.

B. APPROXIMATION AND EXACT STRATEGIES

After Leiden community detection, the transaction database is decomposed into several highly related transaction communities. Unlike traditional mining of an entire database, we separately mine each transaction community and propose two strategies: approximation and exact strategies. These strategies can be used for mining based on specific needs. Nevertheless, unless otherwise stated, we use the exact strategy as the general approach. In the approximation strategy, each transaction community is processed without considering shared items. Initially, local frequent patterns are extracted from each transaction community, and a merging function is then employed to derive global frequent patterns. This function comprises the concatenation of all local frequent patterns. The advantage of this strategy is fast mining, but the downside is that it does not consider shared items, resulting in an incomplete mining process. Algorithm 1 is a pseudocode for the approximation strategy.

```
Algorithm 1 Approximation Strategy
Input: $k$ transaction communities $C = \{C_1, C_2, \ldots, C_k\}$; minimum support min$Sup$;
Output: $FIs$.
```

Fig. 2 shows the workflow of the approximation strategy. In the exact strategy, each transaction community is processed individually and shared items between different transaction communities are considered. The goal of this strategy is to capture frequent patterns not covered by local

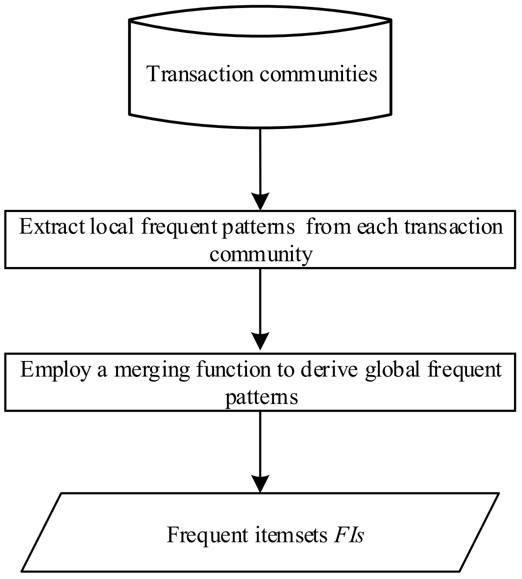
Transaction communities
Extract local frequent patterns from each transaction community
Employ a merging function to derive global frequent patterns
Frequent itemsets FIs

FIGURE 2. Algorithm 1: approximation strategy.

transaction communities, thereby achieving the mining of all frequent patterns. Initially, local frequent patterns are extracted from each transaction community, and then the potential frequent patterns generated by shared items between different transaction communities are considered. Finally, a merging function is employed to derive global frequent patterns. The advantage of this strategy is that it can mine all frequent patterns, but it is relatively slower due to the consideration of shared items. Algorithm 2 is a pseudocode for the exact strategy.

```
Algorithm 2 Exact Strategy
Input: $k$ transaction communities $C = \{C_1, C_2, \ldots, C_k\}$; minimum support min$Sup$; shared item set $S$.
Output: $FIs$.
```

Fig. 3 shows the workflow of the exact strategy.

C. OVERALL DESIGN AND EXAMPLE DEMONSTRATION OF LcGA-FCHUIM

Algorithm 3 is the pseudocode for LcGA-FCHUIM. The algorithm's goal is to efficiently find as many *FCHUIs* as possible that meet certain criteria, but it does not guarantee to find all of them. The algorithm proceeds as follows.

Use Leiden community detection to decompose the transaction dataset into several categories. Use an approximation or exact strategy to mine FIs. Apply closure detection to FIs to obtain frequent closed itemsets. Use cGA to mine HUIs within the frequent closed itemsets.

Fig. 4 shows the workflow of LcGA-FCHUIM. For a clearer demonstration of the algorithm, we consider an example. The transaction database is shown in Table 1. After using Leiden community detection, the transactions are divided into three classes (Fig. 5).

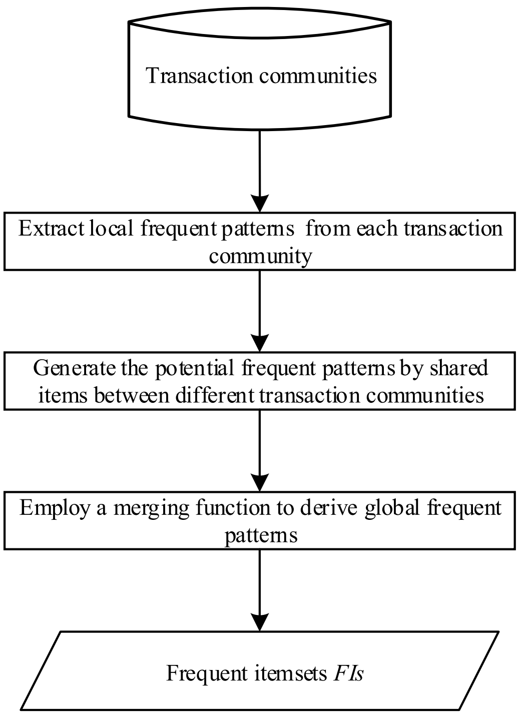
Transaction communities
Extract local frequent patterns from each transaction community
Generate the potential frequent patterns by shared items between different transaction communities
Employ a merging function to derive global frequent patterns
Frequent itemsets *FIs*

FIGURE 3. Algorithm 2: exact strategy.

Algorithm 3 LcGA-FCHUIM

Input: Transaction dataset D; minimum support *minSup*; minimum utility threshold min*Uti*; fitness function $f(c) = u(X)$.

Output: *FCHUIs*.

$C_1 = \{T_1, T_5\}$, $I(C_1) = \{a, c, d, e\}$; $C_2 = \{T_4, T_6\}$, $I(C_2) = \{a, b, f\}$; $C_3 = \{T_2, T_3\}$, $I(C_3) = \{c, d, e\}$, where $I(C_i)$ represents the set of all items between classes. The shared itemset between C_1 and C_2 is $\{a\}$, that between C_1 and C_3 is $\{c, d, e\}$, and there are no shared items between C_2 and C_3. Shared items are indicated above the diagonal line. Next, we use the approximation and exact strategies to mine the transaction communities separately. Using a support threshold of 33%, we obtain $L_1 = \{a, c, e, ac, ae, ce, ace\}$, $L_2 = \{b, f, bf\}, L_3 = \{e\}$, where L_i represents the FIs mined from each class. For the approximation strategy, $FIs = L_1 \cup L_2 \cup L_3 = \{a, b, c, e, f, ac, ae, ce, bf, ace\}$. For the exact strategy, we consider the FIs obtained using the approximation strategy and possible combinations of shared items. Thus, $L = L_1 \cup L_2 \cup L_3 \cup \{a, c, d, e, ce, de\} = \{a, b, c, d, e, f, ac, ae, ce, de, bf, ace\}$. With a support threshold of 50%, $L_1 = L_2 = L_3 = \emptyset$ for the approximation strategy, and $L = L_1 \cup L_2 \cup L_3 = \emptyset$. Meanwhile, for the exact strategy, $L = L_1 \cup L_2 \cup L_3 \cup \{a, c, e, ce\} = \{a, c, e, ce\}$. When the support is relatively low, the approximation and exact strategies yield similar FIs. However, when the support is relatively high, there are significant differences in the FIs obtained. In addition, the approximation strategy is faster and does not require the consideration of shared items. Therefore, for lower support and higher efficiency requirements, the approximation strategy can be selected for mining. Assuming a support of 33% for the exact strategy, closure detection is then applied, $FIs = \{de, bf, ace\}$. After closure detection, the cGA process for HUIM begins. First, the probability vector is initialized, and $P = \{0.5,0.5,0.5,0.5,0.5,0.5\}$. For example, if *de* and *bf* are chosen initially, their encodings are 000110 and 010001, and their fitness values are 32 and 23, respectively. Comparing these, the winner is 000110. Assuming min*Uti* = 30, *de* becomes an FCHUI. Next, the probability updating process occurs. Because the encodings are not the same at each position (000110 and 010001), the probability vector is updated based on the winner's position. Thus, we have $P = (0.33,0.33,0.33,0.67,0.67,0.33)$. Because the termination condition is not met, the algorithm returns to Step 6 and continues. In the end, the algorithm returns $FCHUIs = \{de, ace\}$.

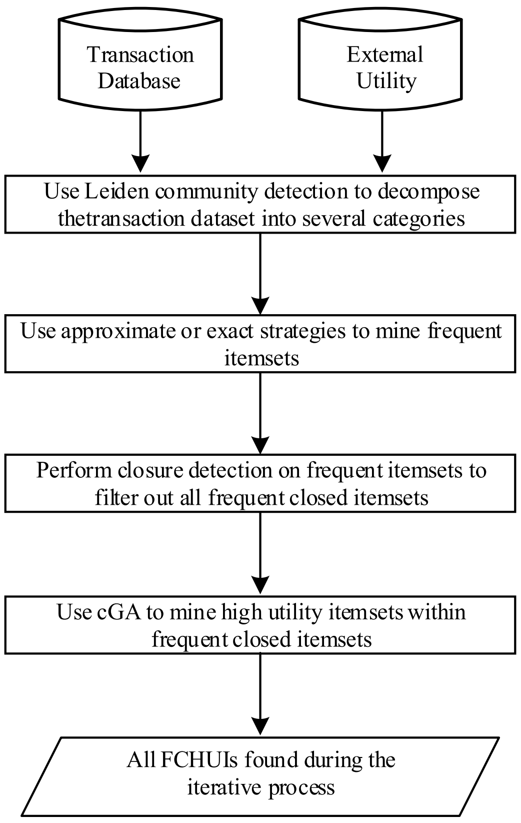
Transaction Database
External Utility
Use Leiden community detection to decompose thetransaction dataset into several categories
Use approximate or exact strategies to mine frequent itemsets
Perform closure detection on frequent itemsets to filter out all frequent closed itemsets
Use cGA to mine high utility itemsets within frequent closed itemsets
All FCHUIs found during the iterative process

FIGURE 4. Algorithm 3: LcGA-FCHUIM.

D. COMPLEXITY OF LcGA-FCHUIM

In this subsection, we analyze the complexity of the proposed LcGA-FCHUIM algorithm by considering each main step of the algorithm. Assuming that the dataset to be mined consists of m transactions and n items, after applying Leiden community detection, the dataset is divided into k communities.

1. Using Leiden Community Detection to Decompose the Dataset The complexity of the Leiden community detection

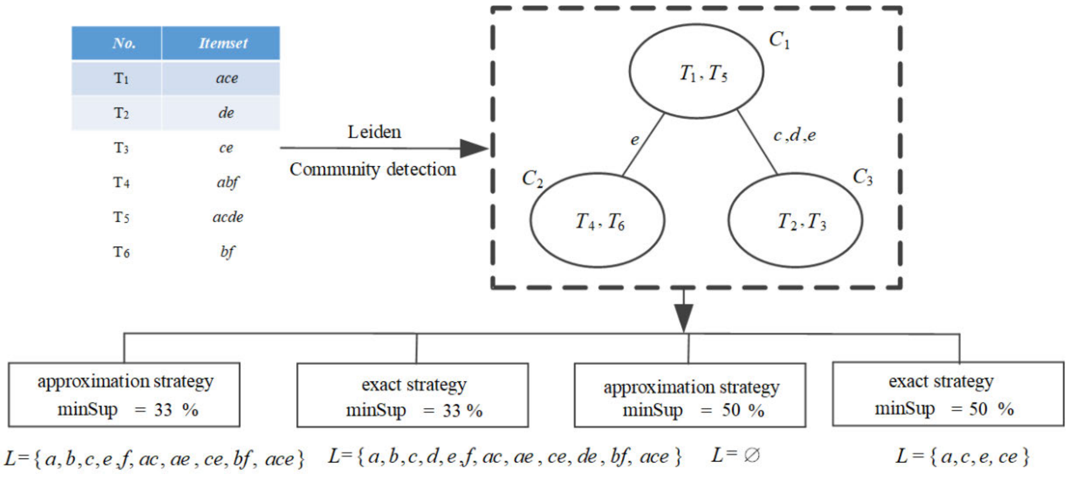
No.
Itemset
T1 ace
T2 de
T3 ce
T4 abf
T5 acde
T6 bf
Leiden
Community detection
C1
C2
C3
T1, T5
T4, T6
T2, T3
e
c, d, e
approximation strategy minSup = 33 %
exact strategy minSup = 33 %
approximation strategy minSup = 50 %
exact strategy minSup = 50 %
L={a,b,c,e,f, ac, ae, ce, bf, ace}
L={a,b,c,d,e,f, ac, ae, ce, de, bf, ace}
L=∅
L={a,c,e, ce}

FIGURE 5. Example demonstration.

algorithm depends mainly on the number of nodes and edges in the graph. Assuming that the transaction dataset is represented as a graph in which nodes represent items and edges represent co-occurrence relationships between items, for a graph with n items, the time complexity of Leiden community detection is $O(m \times n)$.

2. Mining Frequent Itemsets In this step, we use the currently optimal frequent itemset mining algorithm PrePost + [24], which itself has a computational complexity of $O(n\log n)$. Assuming that we adopt an approximation strategy, we do not need to consider the items shared among communities. Thus, the final computational complexity of this step is $O((n\log n)/k+k^2)$. If we use an exact strategy, the items shared among communities must be considered, and in this case, the final computational complexity of this step is $O((n\log n)/k)$.

3. Closure Detection Closure detection involves checking whether each frequent itemset is closed. For l frequent itemsets, the complexity of closure detection is $O(m \times l^2)$ because it requires us to compare each frequent itemset with others and calculate their supports.

4. Mining High Utility Itemsets The cGA is used to mine high-utility itemsets, assuming an initial population size of p and t iterations. Here, each utility function evaluation has a complexity of $O(m)$, and hence, the complexity of cGA is $O(p \times t \times m)$.

5. Total Time Complexity The overall time complexity under the approximate strategy is:

$$O(m \times n) + O((n \log n)/k) + O(m \times l^2) + O(p \times t \times m)$$

The overall time complexity under the exact strategy is:

$$O(m \times n)+O((n \log n)/k+k^2)+O(m \times l^2)+O(p \times t \times m)$$

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we compare the performance of LcGA-FCHUIM with those of state-of-the-art CHUIM algorithms, specifically CLS-Miner [10] and CHUI-Miner [9]. Their modified versions that meet the requirements of FCHUIM are denoted as CLS-Miner-S and CHUI-Miner-S, respectively. We will evaluate the performance in terms of runtime, convergence, and the number of discovered FCHUIs.

A. EXPERIMENTAL ENVIRONMENT AND DATASETS

The experiments were performed on a computer with the following configuration: Intel Core i5-6402P CPU (2.8GHz), 16GB of RAM, and a 64-bit Windows 7 operating system, with Java as the programming language. Four datasets, Retail, Chainstore, Chess, and Accidents, were used for comparative experiments. All four datasets are from the open-source SPMF community [25]. The datasets are described below.

Retail is a medium to large sparse dataset which records consumer purchase data in a mall. Chainstore is a large sparse dataset which records customer transaction data in a large retail chain. Chess is a small dense dataset which records game moves in chess. Accidents is a medium dense dataset which records accident data, such as road accidents. Because of the variations in the performance of different algorithms on datasets of different sizes and sparsity, we chose these four datasets to test the performance of the proposed algorithm from various perspectives.The features of these datasets, such as average transaction length, number of items, number of transactions, and data type, are summarized in Table 3. The number of transactions reflects the size of the dataset, number of items reflects the size of the solution space to some extent, and the average length reflects the sparseness of the dataset to some extent.

B. RUNTIME COMPARISON

In this study, we compare LcGA-FCHUIM with CLS-Miner-S and CHUI-Miner-S in two scenarios. The first scenario involves fixing minSup = 0 and comparing the runtime for various datasets for different min$Util$ values. This scenario represents CHUIM, and the results are shown in Fig. 3. The second scenario involves fixing min$Util$ values for

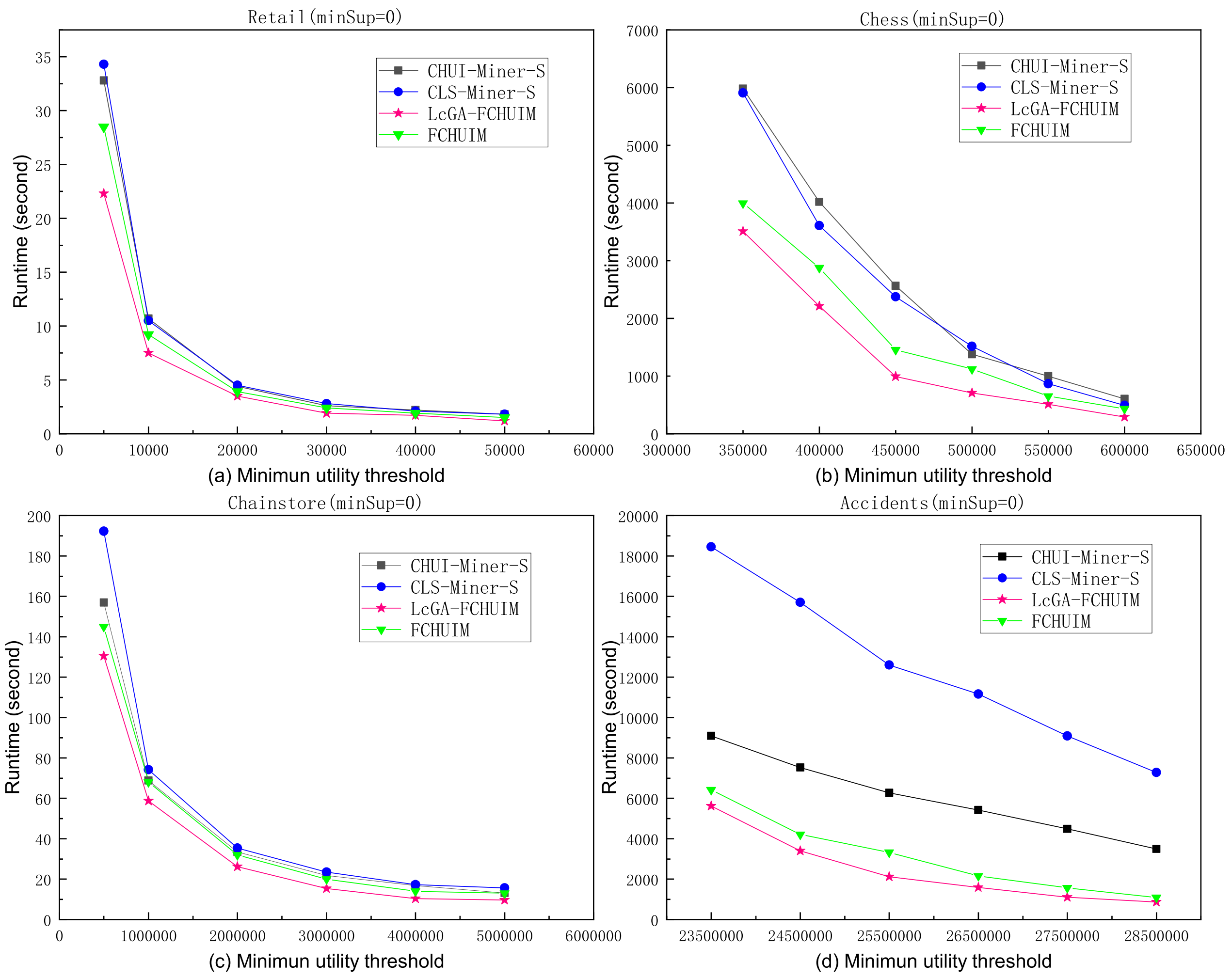

FIGURE 6. **Comparison of running times at minSup = 0 on different datasets.**

TABLE 3. **The features of datasets.**

DataSet	Average length	Number of items	Number of transactions	Datatype
Retail	10.3	16470	88162	Sparse
Chainstore	7.23	46086	1112949	Sparse
Chess	37	75	3196	Dense
Accidents	33.8	468	340183	Dense

each dataset and comparing runtimes for different minimum support thresholds. The results for this scenario are shown in Fig. 4.

Fig. 3 shows that LcGA-FCHUIM outperforms other algorithms in terms of runtime across all four datasets, attributable to the Leiden community detection, which divides the datasets into highly related communities. cGA can efficiently search for more concise related patterns within these communities, significantly reducing the search space and improving efficiency. In addition, cGA is inherently an efficient probabilistic search algorithm and can handle high-dimensional variables, further enhancing LcGA-FCHUIM's runtime efficiency. Notably, for sparse datasets, the runtime decreases significantly at low min*Sup* values, indicating that there are few itemsets with substantial support. Meanwhile, for dense datasets, a substantial decrease in runtime occurs when min*Sup* reaches 30%, suggesting that dense datasets contain more itemsets with substantial support. Figure 6 shows that the proposed algorithm does not have a significant runtime advantage over the comparison algorithms on sparse datasets such as Retail and Chainstore. However, because of its structure, the proposed algorithm has a clear advantage on dense datasets such as Chess and Accident. Algorithms based on linked list structures are sensitive to the dataset density. Although they can quickly prune sparse datasets, they require more search time for dense datasets. In contrast, probability-based search algorithms are insensitive to the density of the dataset and tend to perform better on dense datasets.

C. CONVERGENCE COMPARISON

Fig. 8 shows the convergence of two variants of our cGA-based algorithm. One variant, LcGA-FCHUIM,

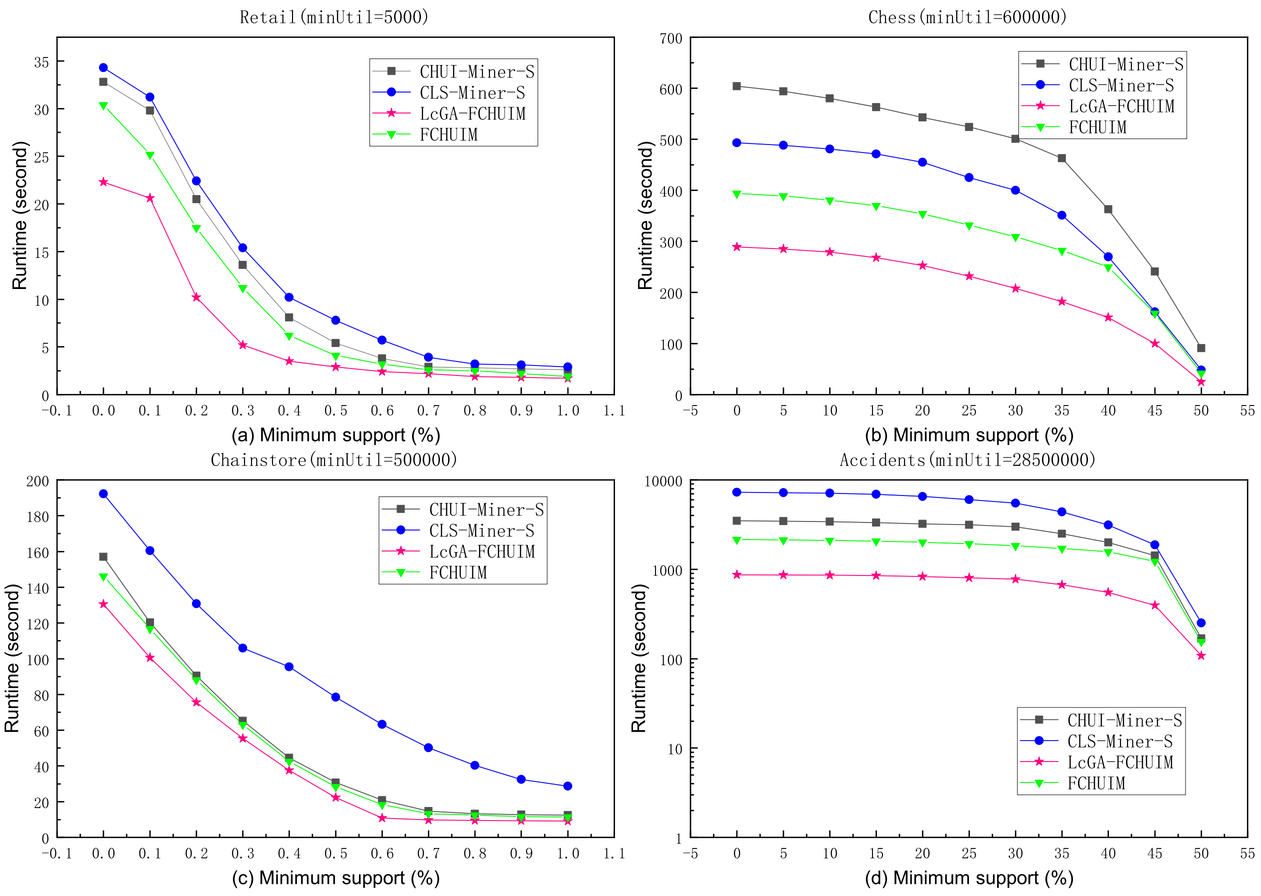

FIGURE 7. Comparison of running times when minSup changes on different datasets.

incorporates Leiden community detection, whereas the other variant, cGA-FCHUIM, does not. LcGA-FCHUIM converges slightly faster and discovers more FCHUIs than cGA-FCHUIM, approximately 96% and 77% of FCHUIs, respectively. This difference is due to the highly related nature of the data chunks obtained after Leiden community detection. cGA can find more FCHUIs within the communities, supporting the effectiveness of introducing Leiden community detection for data classification before pattern mining. However, the convergence speed did not improve significantly, possibly because LcGA-FCHUIM only has an additional step of data preprocessing compared to cGA-FCHUIM. The former can mine a considerable number of HUIs faster in the early stages, whereas in the later stages, the exploration mode of the solution space for both algorithms is guided by a probabilistic model, and there is no significant difference.

D. COMPARISON OF DISCOVERED FCHUI QUANTITY

Table 4 presents a comparison of the quantity of discovered FCHUIs using five algorithms acrxoss the four datasets. The algorithms are as follows.

CLS-Miner-S and CHUI-Miner-S are exact algorithms; they can discover 100% of FCHUIs. The other three algorithms are variants of our proposed algorithm. LcGA-FCHUIM, although not exact, can discover an average of 96% of FCHUIs, making it suitable for most scenarios. LcGA-FCHUIM-AP and cGA-FCHUIM-AC, which experience more substantial losses in discovering FCHUIs, average approximately 79.5% and 77.5%, respectively. The loss in LcGA-FCHUIM-AP arises from not considering that shared items can generate FCHUIs when using the approximation strategy. The loss in cGA-FCHUIM-AC is because is cGA not an exact algorithm, leading to lower search efficiency for unprocessed datasets. From another perspective, this reflects the effectiveness of combining Leiden community detection with cGA to mine FCHUIs.

LcGA-FCHUIM-AP is suitable for scenarios where precision is not critical but short execution times are. For instance, in environments with limited computational resources, such as mobile devices or embedded systems, the algorithm can perform efficient data mining tasks without occupying a large amount of computational resources [26]. Similarly, when facing large-scale datasets, the algorithm can provide a rough analysis result within a reasonable timeframe, helping analysts determine which aspects are worth conducting more in-depth research on.

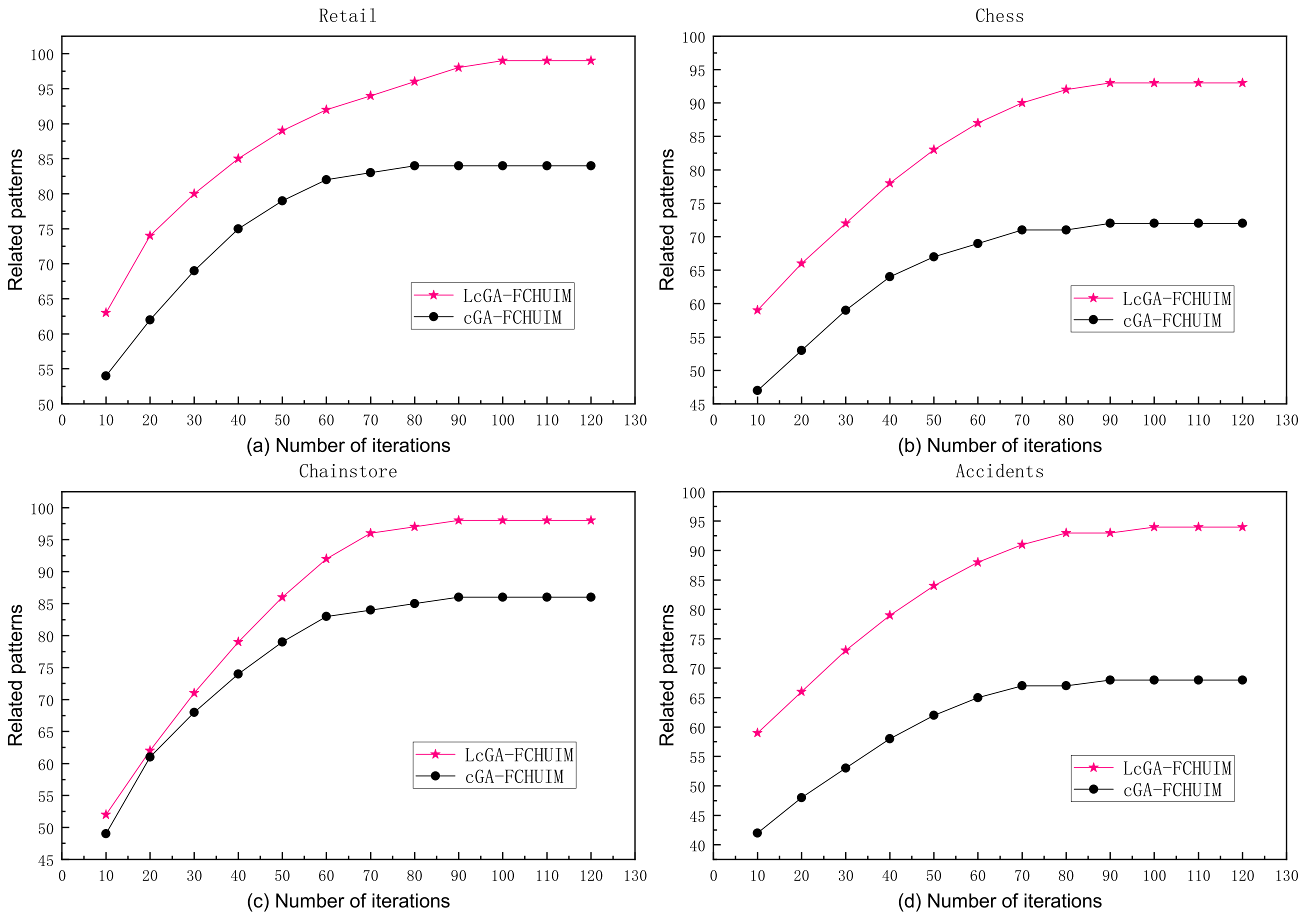

FIGURE 8. Convergence on different datasets.

TABLE 4. Comparison of the mining quantity of each algorithm on different datasets.

DataSet	LcGA-FCHUIM-AC	LcGA-FCHUIM-AP	cGA-FCHUIM-AC	CHUI-Miner-S	CLS-Miner-S	FCHUIM
Retail	99.42%	84.12%	82.32%	100%	100%	100%
Chainstore	98.23%	86.33%	88.56%	100%	100%	100%
Chess	93.33%	72.16%	75.48%	100%	100%	100%
Accidents	94.58%	68.58%	73.33%	100%	100%	100%

VI. CONCLUSION

In this study, we proposed a frequent closed HUIM algorithm based on the Leiden community detection algorithm and cGA (LcGA-FCHUIM). The algorithm first uses Leiden community detection to classify a dataset, creating highly related transaction communities. Subsequently, an approximate or exact strategy was used to mine FIs. Closure detection was applied to the FIs, followed by the use of cGA to mine high-utility patterns among frequent closed itemsets. Experiments on four real datasets, both dense and sparse, demonstrated that LcGA-FCHUIM exhibited the highest runtime efficiency. Although the proposed mining algorithm is not exact, it can discover an average of 96% of FCHUIs, making it suitable for most scenarios. However, we still note that on sparse datasets, the proposed algorithm does not have a significant advantage in terms of runtime over other algorithms. Nonetheless, on dense datasets, it is advantageous in terms of efficiency because of its structure. Mining algorithms based on linked list structures can rapidly prune sparse spaces but require more search time for dense spaces. In contrast, probability-based search algorithms are insensitive to dataset density and perform relatively better on dense datasets. Future work will explore models that combine machine learning and evolutionary algorithms to mine utility patterns, further enhancing efficiency. Moreover, parallel algorithms based on Spark are worth investigating.

ACKNOWLEDGMENT
The authors would like to express their gratitude to Dr. Fei Ma, Dr. Wei Lian, and Dr. Zihan Guo for their participation and valuable contributions. They also grateful to the Institute of Artificial Intelligence at Changzhi University for providing them with the research site and necessary equipment. It is important to note that the findings and opinions presented in this article solely represent the views of the authors and may not necessarily align with those of the sponsors of this research.

REFERENCES
[1] W. Gan, J. C. Lin, P. Fournier-Viger, H.-C. Chao, V. S. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1306–1327, Apr. 2021.
[2] J. Miao, S. Wan, W. Gan, J. Sun, and J. Chen, "Targeted high-utility itemset querying," *IEEE Trans. Artif. Intell.*, vol. 4, no. 4, pp. 871–883, Aug. 2023.
[3] P. Fournier-Viger, J. Chun-Wei Lin, T. Truong-Chi, and R. Nkambou, "A survey of high utility itemset mining," in *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Cham, Switzerland: Springer, 2019, pp. 1–45.
[4] Z.-H. Deng, "An efficient structure for fast mining high utility itemsets," *Appl. Intell.*, vol. 48, pp. 3161–3177, May 2018, doi: 10.1007/s10489-017-1130-x.
[5] S. Kannimuthu and K. Premalatha, "Discovery of high utility itemsets using genetic algorithm with ranked mutation," *Appl. Artif. Intell.*, vol. 28, no. 4, pp. 337–359, Apr. 2014, doi: 10.1080/08839514.2014.891839.
[6] K. Lodhi, "Survey on frequent pattern mining," *Int. J. Eng., Sci. Math.*, vol. 2, no. 3, pp. 64–77, Dec. 2013.
[7] P. Fournier-Viger, S. Zida, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "EFIM-closed: Fast and memory efficient discovery of closed high-utility itemsets," in *Proc. 12th Int. Conf.* New York, NY, USA: Springer, 2016, pp. 199–213.
[8] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining the concise and lossless representation of high utility itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 726–739, Mar. 2015, doi: 10.1109/TKDE.2014.2345377.
[9] J. C.-W. Lin, Y. Djenouri, G. Srivastava, and J. M.-T. Wu, "Large-scale closed high-utility itemset mining," in *Proc. Int. Conf. Data Mining Workshops (ICDMW)*, 2021, pp. 591–598.
[10] T.-L. Dam, K. Li, P. Fournier-Viger, and Q.-H. Duong, "CLS-miner: Efficient and effective closed high-utility itemset mining," *Frontiers Comput. Sci.*, vol. 13, no. 2, pp. 357–381, Apr. 2019, doi: 10.1007/s11704-016-6245-4.
[11] W. Gan, Z. Du, W. Ding, C. Zhang, and H.-C. Chao, "Explainable fuzzy utility mining on sequences," *IEEE Trans. Fuzzy Syst.*, vol. 29, no. 12, pp. 3620–3634, Dec. 2021.
[12] J. C.-W. Lin, Y. Djenouri, G. Srivastava, U. Yun, and P. Fournier-Viger, "A predictive GA-based model for closed high-utility itemset mining," *Appl. Soft Comput.*, vol. 108, Sep. 2021, Art. no. 107422, doi: 10.1016/j.asoc.2021.107422.
[13] F. Zhang, M. Liu, F. Gui, W. Shen, A. Shami, and Y. Ma, "A distributed frequent itemset mining algorithm using spark for big data analytics," *Cluster Comput.*, vol. 18, no. 4, pp. 1493–1501, Dec. 2015, doi: 10.1007/s10586-015-0477-1.
[14] T. Wei, B. Wang, Y. Zhang, K. Hu, Y. Yao, and H. Liu, "FCHUIM: Efficient frequent and closed high-utility itemsets mining," *IEEE Access*, vol. 8, pp. 109928–109939, 2020, doi: 10.1109/ACCESS.2020.3001975.
[15] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mechanics: Theory Exp.*, vol. 2008, no. 10, Oct. 2008, Art. no. P10008, doi: 10.1088/1742-5468/2008/10/p10008.
[16] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to leiden: Guaranteeing well-connected communities," *Sci. Rep.*, vol. 9, no. 1, p. 5233, Mar. 2019, doi: 10.1038/s41598-019-41695-z.
[17] T. Ryu et al., "Scalable and efficient approach for high temporal fuzzy utility pattern mining," *IEEE Trans. Cybern.*, vol. 53, no. 12, pp. 7672–7685, Dec. 2023.
[18] Y. Djenouri, J. C.-W. Lin, K. Nørvåg, H. Ramampiaro, and P. S. Yu, "Exploring decomposition for solving pattern mining problems," *ACM Trans. Manage. Inf. Syst.*, vol. 12, no. 2, pp. 1–36, Jun. 2021, doi: 10.1145/3439771.
[19] C.-W. Wu, P. Fournier-Viger, J.-Y. Gu, and V. S. Tseng, "Mining compact high utility itemsets without candidate generation," in *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Cham, Switzerland: Springer, 2019, pp. 279–302.
[20] C. Jiang and X. Xue, "A uniform compact genetic algorithm for matching bibliographic ontologies," *Appl. Intell.*, vol. 51, pp. 7517–7532, Aug. 2021, doi: doi.org/10.1007/s10489-021-02208-6.
[21] B. Doerr, "The runtime of the compact genetic algorithm on jump functions," *Algorithmica*, vol. 83, no. 10, pp. 3059–3107, Oct. 2021, doi: 10.1007/s00453-020-00780-w.
[22] K. Dehghanpour, Z. Wang, J. Wang, Y. Yuan, and F. Bu, "A survey on state estimation techniques and challenges in smart distribution systems," *IEEE Trans. Smart Grid*, vol. 10, no. 2, pp. 2312–2322, Mar. 2019.
[23] J. C.-W. Lin, Y. Djenouri, G. Srivastava, and P. Fourier-Viger, "Efficient evolutionary computation model of closed high-utility itemset mining," *Appl. Intell.*, vol. 52, no. 9, pp. 10604–10616, Aug. 2022.
[24] Z.-H. Deng and S.-L. Lv, "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning," *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5424–5432, Aug. 2015, doi: 10.1016/j.eswa.2015.03.004.
[25] P. Fournier-Viger, "The SPMF open-source data mining library version 2," in *Proc. Eur. Conf.*, Riva del Garda, Italy. Cham, Switzerland: Springer, Sep. 2016, pp. 36–40.
[26] Y. Baek, U. Yun, H. Kim, J. Kim, B. Vo, T. Truong, and Z.-H. Deng, "Approximate high utility itemset mining in noisy environments," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106596.

XIUMEI ZHAO received the bachelor's degree in engineering from Shanxi Normal University, in 1997, and the master's degree in engineering (computer science, technology, and applications) from Shanxi University, in 2005. Her research and teaching include software engineering, data mining, and more, and she has published over 20 journals in related teaching and research fields.

XINCHENG ZHONG received the master's degree in pattern recognition and Intelligent systems, university of Chinese Academy of Sciences, China, in 2016. From June 2016 to September 2023, he was a lecturer at Changzhi University, China. From September 2023 to the present, he is a visiting scholar at the College of Computer and Information Technology, Shanxi University, China. His primary research interests include artificial intelligence, data mining.

BING HAN received the M.S. degree in mapping and geographic information system from Nanjing Normal University. She was a Visiting Scholar at North Arizona University, from August 2017 to December 2017. She is currently a Lecturer with Changzhi University. Her primary research interests include deep learning and NLP.