Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6653816, 12 pages
https://doi.org/10.1155/2021/6653816

*Research Article*

# Mining Profitable and Concise Patterns in Large-Scale Internet of Things Environments

**Jerry Chun-Wei Lin**,[1] **Youcef Djenouri**,[2] **Gautam Srivastava**,[3,4] **and Philippe Fournier-Viger**[5]

[1]*Western Norway University of Applied Sciences, Norway*
[2]*SINTEF Digital, Norway*
[3]*Brandon University, Canada*
[4]*China Medical University, Taiwan*
[5]*Shenzhen University, Shenzhen, China*

Correspondence should be addressed to Jerry Chun-Wei Lin; jerrylin@ieee.org

Received 24 December 2020; Accepted 7 September 2021; Published 23 September 2021

Academic Editor: Xingsi Xue



In recent years, HUIM (or a.k.a. high-utility itemset mining) can be seen as investigated in an extensive manner and studied in many applications especially in basket-market analysis and its relevant applications. Since current basket-market scenario also involves IoT equipment to collect information, i.e., sensor or smart devices, it is necessary to consider the mining of HUIs (or a.k.a. high-utility itemsets) in a large-scale database especially with IoT situations. First, a GA-based MapReduce model is presented in this work known as GMR-Miner for mining closed patterns with high utilization in large-scale databases. The $k$ -means model is initially adopted to group transactions regarding their relevant correlation based on the frequency factor. A genetic algorithm (GA) is utilized in the developed MapReduce framework that can be used to explore the potential and possible candidates in a limited time. Also, the developed 3-tier MapReduce model can be easily deployed in Spark for the handlings of any database of large scale for knowledge discovery of closed patterns with high utilization. We created sets of extensive experimental environments for evaluating the results of the developed GMR-Miner compared to the well-known and state-of-the-art CLS-Miner. We present our in-depth results to show that the developed GMR-Miner outperforms CLS-Miner in many criteria, i.e., memory usage, scalability, and runtime.

## 1. Introduction

As there is rapid growth of information technologies regarding machine learning models, Internet of Things (IoT) [1], and edge and cloud computing [2, 3], data-driven mining has become an important topic that can be used to extract the meaningful information from the collections of those techniques. Several pattern mining models [4–9] have been extensively studied, and the most fundamental knowledge of pattern mining in knowledge discovery in databases (KDD) is called ARM or association rule mining, which is deployed through varied applications and specific domains. Among them, Apriori was presented for finding the association rules set in transactional databases iteratively. This is a standard approach that finds the candidate itemsets first then derive the satisfied itemsets at each level or called as the level-wise/generate-and-test model; a huge memory usage and the computational cost are relevantly high. After that, a set of association rules can be discovered and mined. Frequent pattern- (FP-) tree [10] was designed to speed up mining progress by building a condense tree structure. Thus, only frequent 1-itemsets are held in the main memory that can be used for later mining progress. In addition, a conditional FP-tree is then recursively constructed to find the frequent itemsets (or frequent patterns) according to different prefix itemsets in the Header_Table. Both Apriori and FP-tree algorithms ensure the DC (or a.k.a. downward closure) property to avoid the heavy cost regarding "combinational explosion."

This property is then applied and extended to many pattern mining algorithms in different domains and applications, i.e., HUIM (or called high-utility itemset mining) [11–15].

HUIM used 2 properties (a.k.a. the internal + external utility) to find the set of HUIs (or a.k.a. high-utility itemsets) in the basket-market domain. The internal utility can be considered as the quantity of an item of each transaction in databases, and external utility can be treated as the unit profit value of each item in databases. Those two values can be replaced by other factors according to the specific requirements, constraints, users' needs, and applications. The generic algorithm of HUIM [16] does not take DC property for revealing the set of HUIs, which requires a huge size of the search space. To solve this limitation, TWU (or a.k.a. transaction-weighted utilization) model [14] considers the transaction utility to construct the HTWUIs (or a.k.a. high transaction-weighted utilization itemsets) as the itemsets with the upper-bound values for maintaining the DC property, which is named as TWDC (or a.k.a. transaction-weighted downward closure) in HUIM. This property is then used in many utility-driven mining algorithms, e.g., UP-growth+ [17], HUI-Miner [15], HUP-tree [18], FHM [19], and d2HUP [20]. More algorithms to improve mining effectiveness regarding the discovered patterns are then developed and discussed by adapting utility concept in pattern mining tasks. In IoT applications [21], many factors can be considered as different values, e.g., interestingness, weight, importance, and uncertainty degree; thus, HUIM can be easily adopted into IoT and/or sensor networks to further discover the required information for data analysis tasks. Based on this assumption, more important and specific information and knowledge will be discovered for later decision or strategy making.

Instead of classic pattern mining approaches such as FIM (or a.k.a. frequent itemset mining) or ARM (or a.k.a. association rule mining) for decision-making, it can disclose more useful and relevant information based on the property of HUIM. The reason is that the HUIM can reveal more information by taking internal and external factors in the mining progress. However, the generic model for discovering the required patterns requires to analyse a huge number of the candidates first, which is inefficient and it is also hard to find the meaningful patterns from a very huge number of patterns. Closed-pattern mining constraint [1, 22–25] was then adapted in pattern mining to provide better functionalities for mining condense and compress patterns. This strategy is then used in HUIM, which is arisen as a new topic called CHUIM (or a.k.a. closed high-utility itemset mining) [26, 27]. Based on this model, less but more meaningful information will be discovered by two conditions as follows: (1) the superset of an itemset has different support values to an itemset itself and (2) the utility of an itemset is no less than the predefined minimum utility count (threshold). CHUD algorithm [26] was investigated to firstly find the CHUIs (or called closed high-utility itemsets) by using the generic TWU model [14]. Since TWU model is a level-wise and generate-and-test model, a huge number of the computational cost is needed and a huge memory usage is required to keep the candidates level-by-level, which is inefficient and time-consuming. CHUI-Miner [27] was investigated to build the extended utility-list (EU-list) that keeps the revealed information in the main memory; the divide-and-conquer mechanism is then used to find the CHUIs correctly and completely. To better improve mining performance, CLS-Miner [28] was designed by using the matrix to lower the size of the search space. This model has good performance compared to the existing models and is considered as the state-of-the-art approach for CHUIM. The generic CLS-Miner is, however, not possible to be performed for discovering the CHUIs in large-scale databases; it is inappropriate in real and industrial domains and applications. Past works have been developed to present the parallel and distributed models used in HUIM [29], but those generic models need to find a very large set of the candidate itemsets for decision-making; it needs high computational cost and a huge memory usage to deliver the complete information. To build an effective and efficient model for revealing the CHUIs has become an important issue in pattern mining research.

Up until now, there has been no model existing that can be used for CHUIM in any database of large scale. Moreover, in the case of correctly and completely mining the needed CHUI making use of distributed and parallel frameworks, we require a strong model to be able to distribute the transactions in an effective and efficient manner to the processing nodes. For solving this known limitation, GMR-Miner is developed and introduced in this paper. Main findings are as follows:

(i) We design a 3-tier MapReduce framework deployed in Spark for mining CHUIs in large-scale datasets

(ii) A $k$-means model is made use of for grouping relevant transactions into clusters; thus, ensuring discovered CHUI numbers is complete and correct

(iii) A GA-based model makes utilization of the MapReduce framework to explore the possible and potential candidates in a limited time for greatly reducing the computational cost

(iv) Experimental evaluation shows that GMR-Miner has a strong and outstanding performance

## 2. Related Work

*2.1. MapReduce Framework.* MapReduce [30] is a parallel and distributed framework that was originally designed and implemented by Dean and Ghemawat. It can be made and implemented to handle large databases. It uses both parallel and distributed models on clusters in 2 main components, Mapper and Reducer, respectively. With regard to pattern mining and the MapReduce framework, the authors in [31] proposed 3 algorithms, using Apriori property to discover the necessary and relevant information. To be used in HUIM, the authors in [29] invented PHUI-growth to be used in the mining of HUIs from big data. As CHUIM research rapidly grows, efficient model development is a necessity for discovering CHUIs in large-scale databases. We refer readers to [29–31] for more in-depth information

on the MapReduce framework and skip an in-depth discussion here in lieu of space considerations in the manuscript.

*2.2. Evolutionary Computation.* Genetic algorithm (GA) was presented by Holland [32] as the first optimization approach in evolutionary computation. The benefit to use GA is that it is not a trivial task to implement GA for real applications. GA is used to solve the NP-hard question and provides a solution optimally. The idea for GA implementation is to encode the solutions as a chromosome, and each chromosome is represented as an individual in the population. To evaluate the goodness of the chromosome, a fitness function should be predefined in the evolutionary process. Since GA is the fundamental approach in evolutionary computation, many extensions [33, 34] are then developed and studied to enhance its efficiency.

In GA, 3 operations are generally considered to iteratively perform for obtaining a better solution, and they are indicated as (1) mutation, (2) crossover, and (3) selection. For the evolutionary progress of GA, first, each possible solution is then encoded as a chromosome, which can be presented as a string by binary or decimal encoding scheme. The crossover operation is then performed to swap the parts of the chromosomes that can be used to produce the offspring as a new solution for the next generation. The idea of crossover operation is to generate the possible solutions and better convergence in a search space. After that, a mutation operation is then executed to flip some digits of a chromosome, which generates new solutions. The idea of mutation operation is to change parts of a solution randomly, which can increase the diversity of the population and provide a mechanism for escaping from a local optimum. Note that the ratio for running the crossover and mutation is different, and normally, the ratio of crossover operation is higher than that of the ratio of mutation operation. After that, the selection operation is then operated to find the elite solutions for the next round (or generation). This selection mechanism is mostly based on the fitness value. Thus, iterative progress is then performed until the termination condition is achieved. Several criteria can be set to terminate the progress of evolutionary model by (1) the number of iterations is achieved by the predefined the number of generations or (2) the fitness value becomes stable without further big changes; the algorithm is converged. However, in traditional GA-based model, it takes long time to be converged by the 3 generic operations.

Several EC-based approaches were adapted to generic ARM [35], HUIM [13, 36], and high average-utility itemset mining (HAUIM) [37] for knowledge discovery. Qodmanan et al. [35] presented a GA-based model to mine the association rules without minimum support and confidence thresholds. The designed fitness function can produce more interesting and important rules rather than the traditional approaches. Kannimuthu and Premalatha [36] first adapted the GA-based model in HUIM that can discover the set of the HUIs in a limit time. Gunawan et al. [13] presented a BSPO model for mining HUIM without threshold value. Further extensions are then developed in progress to adapt the evolutionary computation (EC) for mining the required information. Song and Huang [37] used the PSO model for revealing the high average-utility itemsets.

*2.3. High-Utility Itemset Mining (HUIM).* There can be very beneficial reasons to analyse the purchase behaviours of customers in basket-market domains since the revealed information and knowledge will provide the realistic and profitable values of the products to the company, e.g., supermarket or shopping mall. Generic models of association rule mining/frequent itemset mining only take occurrence frequency as the major consideration, which provide the insufficient knowledge to make the efficient decision especially it is not applicable on an item with lower frequency in the database but can bring higher profit than the others, i.e., diamond or caviar. HUIM [16] was presented to take the internal factor (considered as the quantity of the item in the transactions) and external factor (considered as unit profit for the item in any database) to reveal the set of HUIs, which shows an alternative model for making more precise and accurate strategies for decision-making.

Traditional models of HUIM [16] do not hold the DC property; thus, it takes a very huge search space by "combinational explosion" mechanism to reveal the required information. TWU model [14] was presented to build the upper-bound values on the itemsets by holding and maintaining the HTWUIs. This model can hold the TWDC property to solve the limitation of the past HUIM models. Although TWU model is efficient but it still builds the very high upper-bound values on the itemsets; thus, several models were, respectively, presented to mine the set of HUIs and speed up mining performance. The high-utility pattern- (HUP-) tree was developed to keep the required information into a tree structure, which provides good performance than that of the traditional TWU model. Utility-pattern- (UP-) growth and UP-growth+ [17] were then developed to mine the set of HUIs efficiently from the implemented utility-pattern tree. The above algorithms are, however, still based on TWU model to keep the loose upper-bound values on itemsets; thus, the number of discovered candidates in phase 1 is still a lot. To reduce this limitation by having a lot of candidates in phase 2, HUI-Miner [15] was designed and implemented by a linked-list structure named utility-list- (UL-) structure that can avoid the generate-and-test and tree-based models for mining the set of HUIs. It also uses the join operator to generate $k$-itemsets; thus, the required HUIs at different levels can be found and discovered efficiently. FHM [19] was investigated to build a matrix structure effectively to store the cooccurrence relationships among itemsets that can be used to reduce the search space efficiently since the unpromising candidate itemsets can be early pruned and removed. FIM [38] was then developed and implemented to work on two strategies that can be used to establish the tight upper-bound values on the itemsets; the size of the search space can be reduced greatly. Several works of HUIM are then extensively studied and discussed. Srivastava et al. [39] used the prelarge and fusion models to mine the set of HUIs from wireless sensor networks for the real industry applications. Several approaches and studies are then developed in HUIM, and this research issue has been still developed in progress [9, 40].

Although most of the pattern mining models, e.g., ARM or HUIM, can find the required information for decision-

making, it is sometimes not a trivial task to retrieve the most useful and meaningful information from a huge number of the rules especially for some online decision-making system, i.e., stock market analysis. Thus, it is possible to provide less but meaningful information and knowledge for further decision-making. Closed pattern mining of frequent itemset mining [22, 23] is a good model to find the less but concise patterns as the solution for decision-making. Instead of mining a high number of patterns for decision-making, closed frequent itemset mining can greatly reduce the size of the discovered patterns; thus, it is somehow easier to make the decision in a short time. Closed-pattern mining model was also adapted the concept of HUIM; thus, the CHUI-Miner [27] was presented to find the CHUIs in the databases. Since CHUI-Miner is a one-phase approach; thus, it uses the EU-list model to keep the necessary information for the later mining progress. However, the CHUI-Miner still relies on TWU property to maintain the upper-bound values on the itemsets; it still suffers the limitation of huge search spaces for finding the required patterns; thus, the execution time is costly. Up to now, the state-of-the-art model called CLS-Miner [6] was presented that incorporates the UL-structure EUCS strategy in the mining progress. The EUCS model is very beneficial to reduce the number of 2-itemsets for the further progress; thus, the size of search space can be greatly reduced. Moreover, CLS-Miner applies the efficient strategies to prune the size of the search space as well; thus, the mining performance can be sped up. Up to now, none of the existing models can thus be used to handle the large-scale databases for mining the CHUIs, which is the major task and research issue in this work.

## 3. Preliminary and Problem Statement

A set of items in the database is denoted as $I$ and defined as $I = \{i_1, i_2, \cdots, i_m\}$. Also assume that a database is denoted as $D$ and defined as $D = \{T_1, T_2, \cdots, T_n\}$. Note that each $T_d \subseteq I$ $(1 \le d \le n)$, and there is $n$ transaction in the database $D$. Suppose that the quality of an item $i_j$ in a transaction $T_j$ is denoted as $q(i_j, T_d)$, and the unit profit of an item $i_j$ is denoted as $p(i_j)$. Note that both $q(i_j, T_d)$ and $p(i_j)$ are the positive integers. Assume that an itemset is denoted as $X$ such that $X = \{i_1, i_2, \cdots, i_k\}$. The length of $X$ is considered the size of the itemset $X$, which can be considered as $k$ -itemset $(k = 1, 2, \cdots, m)$. Key definitions of this paper are given as follows.

*Definition 1.* The utility of an item $i_j$ in a transaction $T_d$ is denoted as $u(i_j, T_d)$ and defined as follows:

$$u(i_j, T_d) = q(i_j, T_d) \times p(i_j), \tag{1}$$

where $q(i_j, T_d)$ is the quantitative value of $i_j$ in $T_d$ and $p(i_j)$ is the unit profit of an item $i_j$ in the unit of the profit table.

*Definition 2.* The utility of an itemset $X$ in a transaction $T_d$ is denoted as $u(X, T_d)$ and defined as follows:

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

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

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

*Definition 4.* The utility of a transaction $T_d$ is denoted as $t$ $u(T_d)$ and defined as follows:

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

*Definition 5.* The total utility of a database $D$ is denoted as $u(D)$ and defined as follows:

$$u(D) = \sum_{T_d \in D} tu(T_d). \tag{5}$$

*Definition 6.* Suppose an itemset is defined as $X$, and the minimum utility threshold is set as $\delta$. An itemset is a high-utility itemset (HUI) if it follows the following condition as

$$u(X) \ge \delta \times u(D). \tag{6}$$

*Definition 7.* Suppose an itemset $X$ is a CHUI. It must have the following conditions as follows: (1) any superset (i.e., $Y$) of $X$ will not have the same support value such as $\sup(Y) = \sup(X)$ and (2) $u(X)$ is larger than or equal to the minimum utility count. Note that $u(Y)$ is also larger than or equal to the minimum utility count.

For the generic association rule mining or frequent itemset mining, it holds the downward closure property to avoid the "combinational explosion" issue. To increase the mining performance in HUIM, a new property called transaction-weighted downward closure (TWDC) was established by TWU model [14] that can be used and adapted in HUIM to solve the limitation of the generic models.

*Definition 8.* An itemset is denoted as $X$, and its transaction-weighted utility is denoted as $\mathrm{twu}(X)$. To calculate the transaction-weighted utility of $X$, it follows the condition as follows:

$$\mathrm{twu}(X) = \sum_{T_d \in D \wedge X \subseteq T_d} tu(T_d). \tag{7}$$

Current works [14, 17, 19] regarding HUIM applied the TWU model to keep the TWDC property; it also adapts to CHUIM [27] to avoid the problem of "combinational

explosion." In addition, the UL-list-based model [15] and EUCS-based approach [19] are beneficial to efficiently reveal the required high-utility itemsets. For example, UL-list uses the join operator, which is easily to find the $(k+1)$-itemsets level wisely without candidate generation. The EUCS model uses the matrix structure to keep the TWU values of 2-itemsets. Based on the DC and TWDC properties, if a 2-itemset is not a HTWUI, its superset will not be the HTWUI either; the superset of the itemset can be discarded and ignored. Thus, the search space can be reduced efficiently. As we mentioned, the CHUIM can produce a smaller number of useful and meaningful patterns; thus, it is possible to make the decision quickly based on some specific online applications. The generic models [27, 28] of CCCCCCHUIM cannot, however, handle the large-scale and big datasets, which is not applicable in real-life situations and applications. We thus then developed a MapReduce framework that can be used to process the CHUIM in very big and large-scale datasets.

Problem Statement: Suppose a very large transactional database $D$, and each transaction in $D$ consists of the purchased items with their quantity values. A profit table is assumed as a ptable that keeps the unit profit of the items in the database. Let $\delta$ be the minimum utility threshold in the database. The purpose of this paper is aimed at finding the complete set of the CHUI efficiency by the cloud-computing techniques for handling the large-scale datasets.

## 4. The Developed GA-Based MapReduce Model for CHUIM

We first design a GA-based decomposition model and a 3-tier MapReduce framework for handling large-scale CHUIM in this section. The idea of exploring the decomposition and combining the 3-tier MapReduce is to reduce the search space for finding the required information, which easily is explored by the genetic algorithm (GA). First, the set of transactions $D$ is then partitioned into several groups $G = \{G_1, G_2, \cdots, G_k\}$, in which each group $G_i$ contains several transactions in $D$, and $k$ is set as the group number in the database. Generally, the groups hold disjoint relationship, in which for every two different groups, it holds the condition as follows:

$$\left(G_i, G_j\right), I(G_i) \cap I\left(G_j\right) = \varnothing, \tag{8}$$

where $I(G_i)$ is the set items of the group $G_i$ and $I(G_j)$ is the set items of the group $G_j$.

**Proposition 9.** *Let G be the groups of transactions in the original database D. If the groups in G have no shared items, the set of all relevant frequent itemsets is considered as the unions of the full groups' frequent itemsets. We thus can note that*

$$F = \left\{ \bigcup_{i=1}^{k} F_i \right\}, \tag{9}$$

*where $F_i$ is considered as a set of the relevant frequent itemsets of the group $G_i$.*

*Proof.* Consider $\forall (i, j) \in [1 \cdots k]^2, I(G_i) \cap I(G_j) = \varnothing$, we can obtain that $\forall i \in [1 \cdots k]$: $F_i = \{p \mid \sup(D, I, p) \geq \min\_\sup\}$. The support of the pattern $p$ is examined by checking all transactions in $D$. Considering a pattern $p$ exists in $I(G_i)$, i.e., $p \subseteq I(G_i) \Rightarrow \forall e \in p, e \in I(G_i) \Rightarrow \forall e \in p, e \notin I(G_j), (\forall j \in [1 \cdots k], \vee j \neq i) \Rightarrow p \notin I(G_j) \Rightarrow F_i = \{p \mid \sup(G_i, I(G_i), p) \geq \min\_\sup\} \Rightarrow F = \{\bigcup_{i=1}^{k} F_i\}$.□□ □

The proposition above clearly shows that transactions that are in $D$ must follow certain conditions above, from which the dependent groups can be fully revealed. Thus, relevant frequent itemsets can be identified using pattern mining approaches in groups. However, this is not a realistic scenario, and the objective is to decrease the number of items shared by the separated groups. Existing work [5] identified that $k$-means [41] and DBSCAN [42] can obtain a good performance of transaction decomposition, and $k$-means showed better results than that of the DBSCAN. Thus, a $k$-means model is used in the designed framework for transaction decomposition that can group highly relevant transactions in the same group. After that, a GA-based MapReduce- (GMR-) Miner algorithm that consists of GA and 3-tier MapReduce framework for mining the closed patterns with high utilization is then presented. Three phases in the designed framework regarding different MapReduce tasks are described below.

*4.1. Exploration.* After dividing the transactions into several groups, each Mapper is fed with a partition. The framework for MapReduce is applied in this step for the exploration of any and all promising items which may be CHUI in addition to their supersets. Any unpromising itemsets can easily be discarded in this step to make good mining progress due to the design properties which can be stated as follows.

*Property 10.* We can say that if there exists a known pattern $t$ that clearly is or can be defined as a frequent pattern, it can be defined as a frequent itemset in one part.

*Proof.* Let a database $D$ being split into $n$ parts such that $\{D_1, D_2, \cdots, D_n\}$; the total frequency of each part is calculated as $\{|D_1|, |D_2|, \cdots, |D_n|\}$. Assume that the minimum support threshold is considered as $\delta$ in the database, and $t$ is considered as a frequent pattern in $D$. We then can obtain the following situation as follows:

$$s(t) \geq \delta \times |D_i|. \tag{10}$$

The counter-evidence, $\{s_1, s_2, \cdots, s_n\}$, is used to show the support value of an itemset (pattern) $t$ of each part. Obviously, $t$ is not considered as the frequent itemset in the entire part such that $s_1 < \delta \times |D_1|, s_2 < \delta \times |D_2|, \cdots, s_n < \delta \times |D_n|$. Then, $s(t) = \sum_{i=1}^{n} |D_i|$ is different to the above

definition. This, we can prove that the correctness is held by this property.□□ □

Based on the developed Property 10, it is then studied and extended to the designed MapReduce model. Thus, the integrity of the mined information is then ensured. According to the DC property used in the Apriori algorithm, Property 11 is studied and extended from Property 10 to ensure that the supersets can satisfy the condition. The definition is then given as follows.

*Property 11.* Suppose two itemsets $t$ and $t'$ hold the situation such as $t \subseteq t'$. Thus, in the database $D$, we can ensure that $s(t, D) \geq s(t', D)$ maintains the correctness.

Based on Property 11, if a support of an itemset $t$ is less than the minimum support threshold (count, $\delta \times |D|$), it is not treated as a frequent itemset, neither its supersets. That is, it does not affect the final results if $t$ is then early removed. In the proposed paper, each Mapper acquires a database partition. Thus, the pair of <key, value> for an itemset with its support value (or called frequency) is output in a certain partition to the Reducer. Following that, a GA-based technique is used to investigate the possible search space for the next Reducer phase. All frequent itemsets are treated as individuals in the first population, and then, the unsatisfied itemsets are removed to efficiently minimize the search space for later processes. This GA-based technique can significantly cut computational costs by avoiding the need to explore the whole search space. Following that, all promising frequent itemsets are inspected to determine the complete closed frequent itemsets [22, 23] by the next MapReduce framework to reveal the satisfied CHUIs.

To be concluded, the initial MapReduce divides the clustered dataset into numerous parts (or called partitions), which are subsequently processed independently by each Mapper. The GA model then generates a search space for prospective candidates that can be used to reduce the size of the search space. Following that, all satisfied frequent itemsets are mined and revealed, and unsatisfied frequent itemsets are deleted here. Once again, only satisfied CHUIs will be sent to the subsequent MapReduce model for revealing the set of CHUIs. The following is the description of the exploitation phase.

*4.2. Exploitation.* The exploitation phase begins with the usage of current CHUIM models (e.g., CLS-Miner [28]) to mine the CHUIs for each partition. Given that mining the set of CHUIs in the whole dataset is not straightforward, the second MapReduce is executed in parallel with the partial, tiny, and numerous sets from promising itemsets from the initial MapReduce on each node. Due to the fact that each node requires less memory, the MapReduce architecture is capable of running a large database on a single machine. The candidate's utility is then explored for each node in order to determine the progress of the exploitation mining. The horizontal structure known as tidset is used to store the transaction ID and its associated frequent itemsets. Due to the efficient tidset structure, it is simple to calculate the frequencies of the itemsets in the mining progress; thus, the computational cost can be greatly minimized and the performance can be greatly improved.

Additionally, a straightforward load balancing strategy is used to divide the transactions into the second MapReduce tasks based on their sizes before performing the second MapReduce. The computational cost of the exploitation process can thus be decreased. The number of produced tasks should correspond to the number of Mappers. The workload of each node is determined by the amount of promising itemsets in a transaction, and then, the transaction is assigned to the node with the least workload, which is capable of evenly distributing the computation among the nodes. When compared to the serialization model, this technique can significantly lower processing costs. The load balancing equation is given as follows.

$$\mathrm{WL}_i = \mathrm{WL}_i + \mathrm{Num}, \tag{11}$$

where $\mathrm{WL}_i$ is the workload of node $i$, and Num is the number of patterns derived from the first MapReduce of the performed transaction. The CLS-Miner [28] is applied here to mine the set of local CHUIs at each partition $D_i$. The local CHUI is then output from each Mapper, and the result is a pair of $\{\text{pattern}, (\text{utility}\,; p_i)\}$.

The Mapper stage first executes the CLS-Miner to mine the set of CHUIs within the partition and then assigns the local CHUIs with the same key (or itemset/pattern) to the same Reducer. It is possible to calculate the partial total utility in a partition; the local CHUI can be recognized if its utility value is not smaller than the sum of the partial total utility in the partition. As a result, the CHUI that has been satisfied is output to the result file; otherwise, the Reducers output the key-value pair that will be used later in the generation of the candidate set. Following that, all candidates (possible patterns) and the tidset are required for the next-generation phase, which is completed during the second MapReduce phase. Crossover and mutation procedures are done on the second MapReduce framework to produce the possible candidates for the actual CHUIs between the Mapper and Reducer of each partition.

In this phase, each MapReduce component considers only one cluster of transactions. This allows to highly reduce exploring the solution space. At the same time, the candidate patterns have been calculated for their utilities of each node. Therefore, by using a developed tidset structure, the calculated utility can be used to speed up the checking process.

*4.3. Integration.* The purpose of this stage is to catch any patterns that have been missed in the local clusters due to mining progress. It takes into account both shared and clusters during the exploration and exploitation processes. This enables the discovery of all associated CHUIs across the whole database. From the shared items, potential candidate CHUIs are established initially. It is then investigated to find the significance of each generated pattern over the entire database by utilizing the integration function. The designed framework proposes an aggregation function, which is the

**Input:** $D$, a quantitative database; *ptable*, a profit table of all items; $\delta$ , a minimum utility count.
**Output:** a set of discovered closed high-utility itemsets (CHUIs)
perform $k$-means to cluster $D$ as $(p_1, p_2, \cdots, p_n)$.
perform **exploration** function {
for each $p_k$ {
  set key-value pair as (*tid*, $t$-itemset).
  for each $t$-itemset {
    calculate sup $(t)$.
    }
  write$(t,<$ sup $(t), p_k >)$.
  }
  for each $t$ in $p_k$ {
    sup $(t)$ = sup $(t)$ + <sup $(t), p_k >$ .
    }
  write$(t,$ sup $(t))$.
}
perform **exploitation** {
build *tidset*.
for each $p_k$ {
  set key-value pair as (*tid*, $t$-itemset).
  for each $t$-itemset {
    calculate $u(t)$ by CLS-Miner.
    }
  write$(t,<u(t), p_k >)$
  }
  for each $t$ in $p_k$ {
    $u(t) = u(t) + <u(t), p_k >$ .
    }
  write$(t, u(t))$.
}
perform **integration** {
project $t$-itemset as utility-list (UL).
build EUCS of 2-itemsets.
for each $C$ in $t$ {
  check $Cq$-itemset.
  if $C$ appears in *tidset* and *tid* == key {
    write a pair $(C, lu(C))$.
    else {
    write a pair $(C, lu(C))$.
    }
  }
}
for each $C$ in $t$ {
$gu(C) = gu(C) + lu(C)$.
  if $gu(C) >= \delta \times u(D)$ {
    write$(C,$ gu$(C))$.
  }
}
}

Algorithm 1: The designed GMR-Miner algorithm.

sum of local support for shared patterns across all clusters, to be used as an integration function. Afterwards, the relevant CHUIs of the shared items are concatenated with the relevant CHUIs of the local clusters to derive the globally relevant patterns across the entire transaction database. Additionally, the tidset generated by the second MapReduce is used to decrease the computation required to mine the patterns of each node. Additionally, the utility-list structure

Table 1: The parameters of the used databases.

| Dataset | $\|D\|$ | $\|I\|$ | $C$ | MaxLen |
|---|---|---|---|---|
| SIGN | 730 | 267 | 52 | 94 |
| Leviathan | 5,834 | 9,025 | 33.8 | 100 |
| MSNBC | 31,790 | 17 | 13.3 | 100 |
| BMS | 59,601 | 197 | 2.5 | 267 |

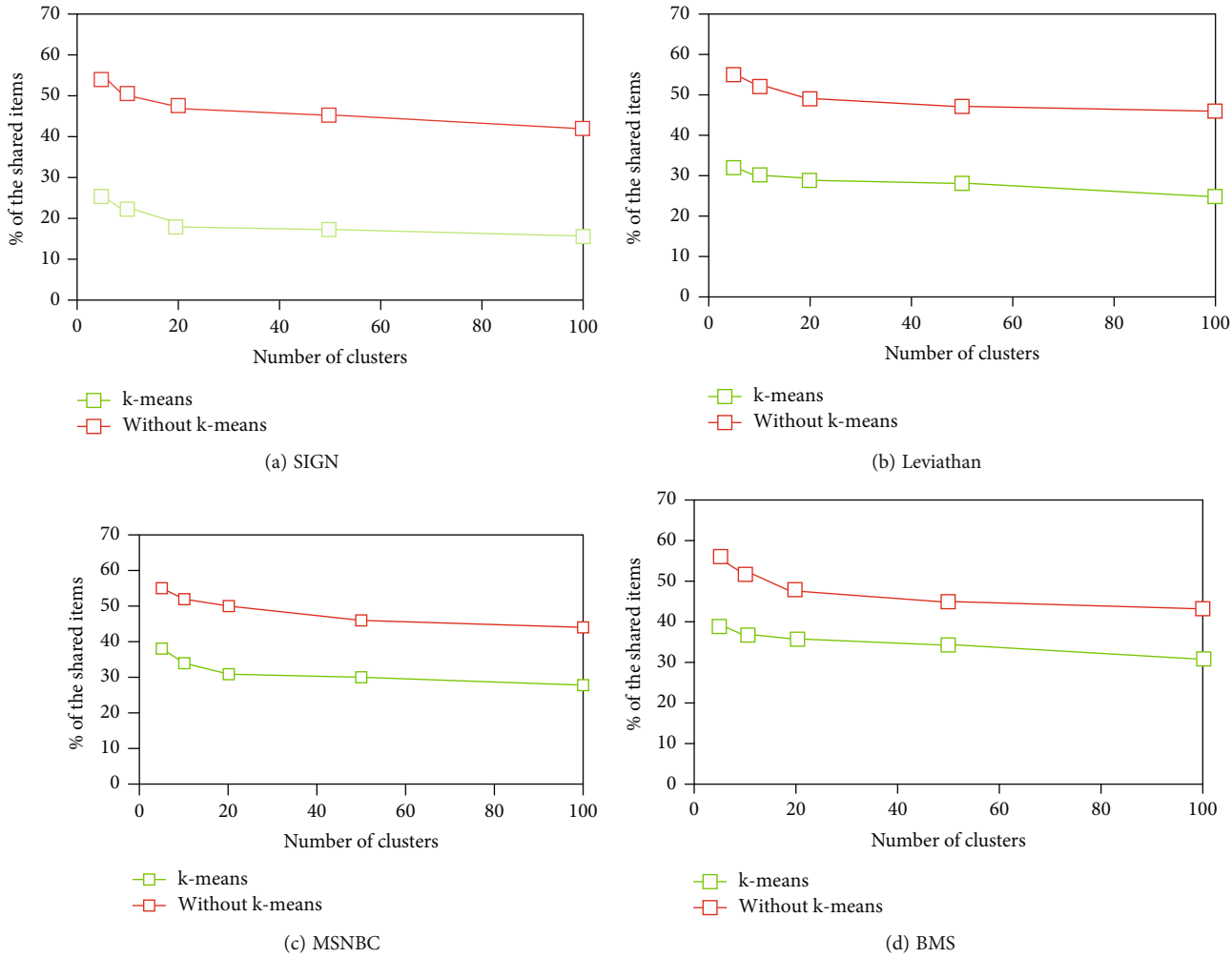



Figure 1: The decomposition clustering quality.

and EUCS are constructed to hold the data required for the calculation, and the computational cost is lowered as a result of these two structures. Additional information on utility-list and EUCS is available in [28].

The third MapReduce framework can then be used to determine and identify global patterns about CHUIs using the set of candidate patterns (local CHUIs) and the tidset. The genetic algorithm's selection operator is used in this phase to retain only the relevant patterns for the next generation, and the fitness (utility) of each candidate pattern is calculated. Each Mapper stage converts the information in the itemset into a utility-list and then determines the local utility of all itemsets in the candidate set. Besides, the EUCS is then applied here to reduce computation if the investigated itemset does not meet the needs. If an investigated itemset can be found from tidset by using its transaction ID, it shows that the utility of the itemset was determined before in the second MapReduce stage; the Mapper here then delivers a pair value of regarding pattern and its utility such as (pattern; utility) for the next Reducer phase. Otherwise, the utility of the pattern can be thus determined by the utility-list structure and a pair value is then output as the result. According to three strategies here such as EUCS, tidset, and utility-list, the mining progress can be sped up, and the computational cost is then reduced for finding the global itemsets with their utility values in the entire database. The Reducer stage here is considered to sum up the utilities of the investigated pattern, and if this value is larger than the $\delta \times u(D)$ in the Reducer stage, it is the globally CHUI and will be released as the final output of the designed framework. Detailed progress of the designed framework is then shown in Algorithm 1.

## 5. Experimental Evaluation

In the experiments, four realistic databases [43] are then used in this paper to state the performance of the developed GMR-Miner approach compared to the state-of-the-art CLS-Miner [28] model in terms of runtime, memory usage, and scalability under a varied number of nodes in the developed 3-tier MapReduce framework. Note that the developed MapReduce is then deployed in Spark since Spark provides a higher capability to handle the large-scale databases. The properties of 4 conducted databases are then described in Table 1. Here, $|D|$ is the number of database size, which showed the number of transactions in the database. $|I|$

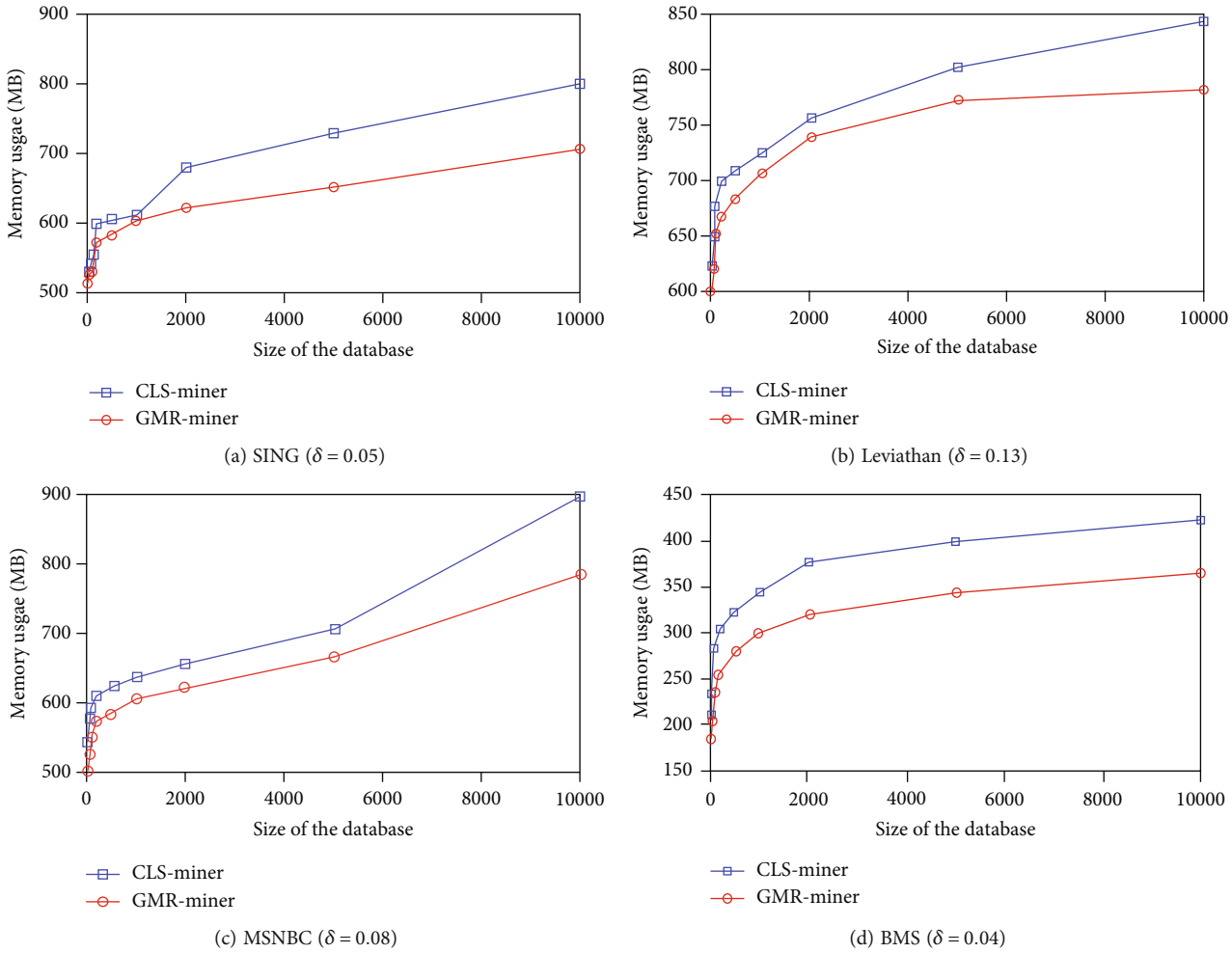
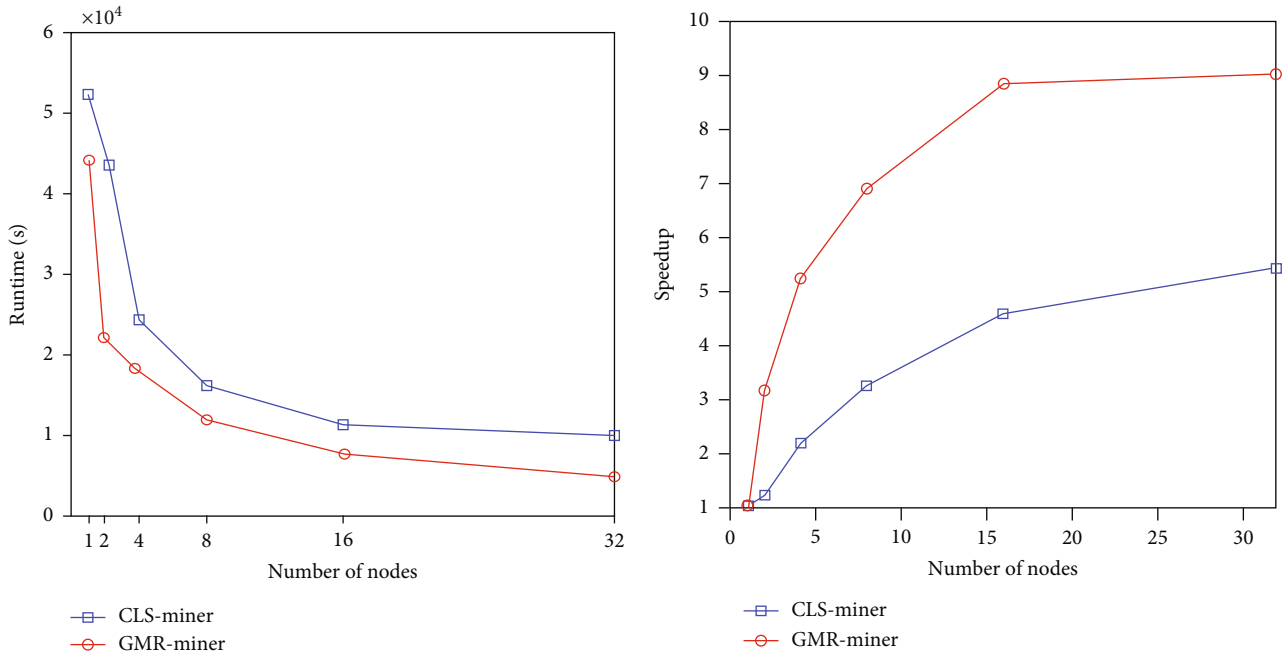



Figure 2: Memory usage of the compared algorithms.

indicated the number of distinct items in the database. $C$ showed the average number of items in a transaction, and MaxLen is the maximum size of a transaction in the database. The used databases in Table 1 are then enlarged and duplicated by various numbers (e.g., 1, 20, 50, 100, 200, 500, 1,000, 2,000, 5,000, and 10,000) for the later performance evaluation.

*5.1. Quality of Clustering.* Figure 1 shows the quality evaluation of the returned clusters by using the $k$-means and the intuitive clustering algorithm on the four datasets used in the experiments. The intuitive clustering divides the transactions into $k$-clusters randomly without any processing. In the conducted experiments, the quality of the returned clusters is then decided by the % of the shared items in clusters, and the object here is to lower the value. We also set the number of clusters in the experiments from 1 to 100; thus, the % for the shared items is then reduced for the evaluation with and without $k$-means approach. However, there is a large difference between $k$-means and intuitive algorithms in all cases. For instance, by using $k$-means to split the transactions, the percentage of shared items does not exceed 40%. However, without using the $k$-means, the percentage of shared items reaches 60%. With the further explanations by the property of $k$-means model, it finds the centroid point based on the similarity equation; the intuitive idea only processes the points by randomness operations. Overall, these experiments clearly showed the benefit of $k$-means in data decomposition. Thus, we can observe that the $k$-means model adapted in this MapReduce framework is useful and effective to mine the CHUIs in large-scale databases.

*5.2. Memory Usage.* To demonstrate the usability of the developed MapReduce model, the results are carried and compared to the CLS-Miner [28] in terms of memory usage, which are shown in Figure 2. By varying the size of the database, it can be seen that the developed GMR-Miner outperforms CLS-Miner in all cases. For instance, only 350 MB is needed by the GMR-Miner to deal with 10,000 times of BMS data. However, 420 MB is needed by the CLS-Miner to handle the same data. These results are reached due to the decomposition step, where each cluster contains similar transactions, and also the intelligent operators of the genetic algorithm where it accurately explores the possible solution space. Thus, less memory usage is then required by the developed GMR-Miner compared to CLS-Miner algorithm.




(a) BMS × 1000 (b) BMS × 1000

Figure 3: Scalability results in terms of runtime and speedup.

Table 2: Clustering quality versus pattern mining accuracy.

| Data | % of the shared items | % of the relevant patters |
|---|---|---|
| SIGN | 40 | 79 |
| | 35 | 88 |
| | 30 | 96 |
| Leviathan | 40 | 77 |
| | 35 | 83 |
| | 30 | 95 |
| MSNBC | 40 | 72 |
| | 35 | 88 |
| | 30 | 97 |
| BMS | 40 | 84 |
| | 35 | 86 |
| | 30 | 91 |

*5.3. Scalability.* To show that the designed GMR-Miner achieves good robustness and applicable in real applications for the large-scale scenario, the scalability on a big dataset is illustrated in Figure 3. Here, we duplicated the BMS dataset 1,000 times for scalability evaluation under a varied number of nodes from 1 to 32. The results showed that the developed GMR-Miner outperforms the CLS-Miner in terms of runtime and speedup under a varied number of nodes, where a high gap between the two approaches is observed. For instance, with 32 nodes, the speedup of the GMR-Miner is 9 for handling 1,000 times of BMS data. However, the speed up of the CLS-Miner is only 5 to handle the same data and with the same number of nodes. This result confirmed the usefulness of genetic algorithms and decomposition for discovering CHUIs in big and large-scale datasets. In general, the developed model can easily process the very big and large databases for mining the required CHUIs, which is very suitable and appropriate for the market engineering.

*5.4. Clustering Quality vs. Pattern Mining Accuracy.* Table 2 presents the quality of the pattern mining process with varying on the clustering quality using the four data (SIGN, Leviathan, MSNBC, and BMS). By varying the quality of the clustering detected by the % of the shared items in the clusters from 40% to 30%, the accuracy of the pattern mining solution increases from 70% to 90% for all the databases used in the experiments. This result is reached thanks to the low dependency among clusters, where the mining process may be applied differently on each cluster of transactions.

## 6. Conclusion and Discussion

Mining high-profitable and concise patterns in IoT environments is not a trivial task since the collected data is usually a large-scale dataset. Past studies of mining CHUIs cannot handle (1) large-scale dataset and (2) mining the required information in a limited time. In this paper, we used a 3-tier MapReduce framework deployed in Spark for efficiently mining the closed patterns with high utilization (or a.k.a. CHUIs). To better explore the possible and potential candidates instead of the entire search space, the genetic algorithm (GA) is also utilized in the designed model for better pattern exploration progress. Experiments are then showed that the designed GMR-Miner outperforms the CLS-Miner in terms of execution time, memory, and scalability regarding a different number of nodes. In the future, a better data

structure can be deployed instead of a utility-list structure for obtaining better performance, and the incremental model can also be investigated and explored as a further research topic to handle the issue of dynamic data mining. In addition, to find the sufficient and satisfied solutions in a limit time, other algorithms such as PSO or ACO in evolutionary computation can also be explored and studied as the further extension.

## Data Availability

The data used to support the findings of this study have been deposited in the SPMF repository (doi:10.1007/978-3-319-46131-1_8).

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

Western Norway University of Applied Sciences, Norway, provides partial funding support for the work carried out in this paper.

## References

[1] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 462–478, 2005.

[2] B. Lin, F. Zhu, J. Zhang et al., "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.

[3] Y. Qu and N. Xiong, "RFH: a resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage," in *2012 41st International Conference on Parallel Processing*, pp. 520–529, Pittsburgh, PA, USA, 2012.

[4] R. Agrawal, T. Imielinski, and A. N. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[5] A. Belhadi, Y. Djenouri, J. C. W. Lin, and A. Cano, "A general-purpose distributed pattern mining system," *Applied Intelligence*, vol. 50, no. 9, pp. 2647–2662, 2020.

[6] 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.

[7] R. U. Kiran, A. Anirudh, C. Saideep, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Finding periodic-frequent patterns in temporal databases using periodic summaries," *Data Science and Pattern Recognition*, vol. 3, no. 2, pp. 24–46, 2019.

[8] H. Si, J. Zhou, Z. Chen et al., "Association rules mining among interests and applications for users on social networks," *IEEE Access*, vol. 7, pp. 116014–116026, 2019.

[9] 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.

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

[11] R. Chan, Q. Yang, and Y. D. Shen, "Mining high utility itemsets," in *IEEE International Conference on Data Mining*, pp. 19–26, Melbourne, FL, USA, 2003.

[12] W. Gan, J. C. W. Lin, P. Fournier-Viger, H. C. Chao, V. Tseng, and P. S. Yu, "A survey of utility-oriented pattern mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, pp. 1306–1327, 2021.

[13] R. Gunawan, E. Winarko, and R. Pulungan, "A BPSO-based method for high-utility itemset mining without minimum utility threshold," *Knowledge-Based Systems*, vol. 190, article 105164, 2020.

[14] Y. Liu, W. Liao, and A. N. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Advances in Knowledge Discovery and Data Mining. PAKDD 2005*, T. B. Ho, D. Cheung, and H. Liu, Eds., vol. 3518 of Lecture Notes in Computer Science, pp. 689–695, Springer, Berlin, Heidelberg, 2005.

[15] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *ACM International Conference on Information and Knowledge Management*, pp. 55–64, Maui, HI, USA, 2012.

[16] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *SIAM International Conference on Data Mining*, pp. 482–486, Lake Buena Vista, Florida, US, 2004.

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

[18] J. C. W. Lin, T. Hong, and W. Lu, "An effective tree structure for mining high utility itemsets," *Expert Systems with Applications*, vol. 38, no. 6, pp. 7419–7424, 2011.

[19] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Foundations of Intelligent Systems. ISMIS 2014*, T. Andreasen, H. Christiansen, J. C. Cubero, and Z. W. Raś, Eds., vol. 8502 of Lecture Notes in Computer Science, pp. 83–92, Springer, Cham, 2014.

[20] J. Liu, K. Wang, and B. C. M. Fung, "Direct discovery of high utility itemsets without candidate generation," in *2012 IEEE 12th International Conference on Data Mining*, pp. 984–989, Brussels, Belgium, 2012.

[21] C. Yin, S. Zhang, J. Wang, and N. N. Xiong, "Anomaly detection based on convolutional recurrent autoencoder for IoT time series," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 21, no. 14, pp. 15626–15634, 2020.

[22] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Information Systems*, vol. 24, no. 1, pp. 25–46, 1999.

[23] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 21–36, 2006.

[24] B. Vo, L. T. T. Nguyen, N. Bui, T. D. D. Nguyen, V. N. Huynh, and T. P. Hong, "An efficient method for mining closed potential high-utility itemsets," *IEEE Access*, vol. 8, pp. 31813–31822, 2020.

[25] 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.

[26] 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 Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 726–739, 2015.

[27] C. W. Wu, P. Fournier-Viger, J. Y. Gu, and V. S. Tseng, "Mining closed+ high utility itemsets without candidate generation," in *2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 187–194, Tainan, Taiwan, 2015.

[28] T. L. Dam, K. Li, P. Fournier-Viger, and Q. H. Duong, "CLS-Miner: efficient and effective closed high-utility itemset mining," *Frontiers of Computer Science*, vol. 13, no. 2, pp. 357–381, 2019.

[29] Y. C. Lin, C. W. Wu, and V. S. Tseng, "Mining high utility itemsets in big data," in *Advances in Knowledge Discovery and Data Mining. PAKDD 2015*, T. Cao, E. P. Lim, Z. H. Zhou, T. B. Ho, D. Cheung, and H. Motoda, Eds., vol. 9078 of Lecture Notes in Computer Science, pp. 649–661, Springer, Cham, 2015.

[30] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] M. Y. Lin, P. Y. Lee, and S. C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," in *The International Conference on Ubiquitous Information Management and Communication*, pp. 1–8, Kuala Lumpur, Malaysia, 2012.

[32] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, 1992.

[33] K. Elbaz, S. L. Shen, A. Zhou, D. J. Yuan, and Y. S. Xu, "Optimization of EPB shield performance with adaptive neuro-fuzzy inference system and genetic algorithm," *Applied Sciences*, vol. 9, no. 4, pp. 780–797, 2019.

[34] R. Guha, M. Ghosh, S. Kapri et al., "Deluge based genetic algorithm for feature selection," *Evolutionary Intelligence*, vol. 14, pp. 357–367, 2021.

[35] H. R. Qodmanan, M. Nasiri, and B. Minaei-Bidgoli, "Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence," *Expert Systems with Applications*, vol. 38, no. 1, pp. 288–298, 2011.

[36] 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.

[37] W. Song and C. Huang, "Mining high average-utility itemsets based on particle swarm optimization," *Data Science and Pattern Recognition*, vol. 4, no. 2, pp. 19–32, 2020.

[38] S. Zida, P. Fournier-Viger, J. C. W. Lin, C. W. Wu, and V. S. Tseng, "EFIM: a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 595–625, 2017.

[39] G. Srivastava, J. C. W. Lin, M. Pirouz, Y. Li, and U. Yun, "A pre-large weighted-fusion system of sensed high-utility patterns," *IEEE Sensors Journal*, 2021.

[40] C. Zhang, G. Almpanidis, W. Wang, and C. Liu, "An empirical evaluation of high utility itemset mining algorithms," *Expert Systems with Applications*, vol. 101, pp. 91–115, 2018.

[41] P. Franti and S. Sieranoja, "How much can k-means be improved by using better initialization and repeats?," *Pattern Recognition*, vol. 93, pp. 95–112, 2019.

[42] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited," *ACM Transactions on Database Systems*, vol. 42, no. 3, pp. 1–21, 2017.

[43] P. Fournier-Viger, J. C. W. Lin, A. Gomariz et al., "The SPMF open-source data mining library version 2," in *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2016*, B. Berendt, Ed., vol. 9853 of Lecture Notes in Computer Science, pp. 36–40, Springer, Cham, 2016.