Information Sciences 553 (2021) 31–48

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework

Jimmy Ming-Tai Wu [a], Gautam Srivastava [b,c], Min Wei [a], Unil Yun [d], Jerry Chun-Wei Lin [e,*]

[a] College of Computer Science and Engineering, Shandong University of Science and Technology, China
[b] Department of Mathematics and Computer Science, Brandon University, 270 18th Street, Brandon R7A 6A9, Canada
[c] Research Center for Interneural Computing, China Medical University, Taichung 40402, Taiwan, ROC
[d] Department of Computer Engineering, Sejong University, Seoul, South Korea
[e] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

ARTICLE INFO

Article history:
Received 10 July 2020
Received in revised form 9 November 2020
Accepted 3 December 2020
Available online 13 December 2020

Keywords:
Hadoop
High fuzzy utility pattern
High utility itemset mining
Big-data
Fuzzy-set theory
MapReduce

ABSTRACT

Over the past decade, high-utility itemset mining (HUIM) has received widespread attention that can emphasize more critical information than was previously possible using frequent itemset mining (FIM). Unfortunately, HUIM is very similar to FIM since the methodology determines itemsets using a binary model based on a pre-defined minimum utility threshold. Additionally, most previous works only focused on single, small datasets in HUIM, which is not realistic to any real-world scenarios today containing big data environments. In this work, the fuzzy-set theory and a MapReduce framework are both utilized to design a novel high fuzzy utility pattern mining algorithm to resolve the above issues. Fuzzy-set theory is first involved and a new algorithm called efficient high fuzzy utility itemset mining (EFUPM) is designed to discover high fuzzy utility patterns from a single machine. Two upper-bounds are then estimated to allow early pruning of unpromising candidates in the search space. To handle the large-scale of big datasets, a Hadoop-based high fuzzy utility pattern mining (HFUPM) algorithm is then developed to discover high fuzzy utility patterns based on the Hadoop framework. Experimental results clearly show that the proposed algorithms perform strongly to mine the required high fuzzy utility patterns whether in a single machine or a large-scale environment compared to the current state-of-the-art approaches.

1. Introduction

With the rapid growth of information techniques, knowledge discovery in databases (KDD) is a critical research field to reveal important, valuable, interesting, and essential knowledge from unprocessed data sources [1–7]. Many different kinds of expertise or rules have been proposed to state the essential concepts from a dataset, and association rule mining (ARM) and/or frequent-itemset mining (FIM) work to gain fundamental knowledge in KDD which has been applied to many domains and applications [4,8–11]. Traditional FIM-based algorithms ignored important factors that are more often than not present in real applications. For example, a commercial company should put more attention to profits and purchase

* Corresponding author.
E-mail addresses: wmt@wmt35.idv.tw (J.M.-T. Wu), srivastavag@brandonu.ca (G. Srivastava), yunei@sejong.ac.kr (U. Yun), jerrylin@ieee.org, jerrylin@ieee.org (J.C.-W. Lin).

https://doi.org/10.1016/j.ins.2020.12.004
0020-0255/

quantities. Unfortunately, FIM does not take into account these types of valuable factors, thus the discovered information may be incomplete or not overly useful.

High-utility itemset mining (HUIM) was proposed to create solutions for revealing more valuable information from a database. HUIM is different from traditional ARM or FIM since it applies utility information (like profit) instead of frequency information for revealing useful patterns. It is usually more suitable to match the requirement of real-world applications. A utility threshold is set by individuals to select high-utility itemsets; if an itemset is called a high-utility itemset, its utility has to be higher than the minimum utility threshold. For example, let us look at an itemset (pattern) as a high-utility itemset in a transaction database. This means that this itemset gained more profit in the past and might have more potential to generate more profit in the future. Therefore, more attention should be given to it. Furthermore, not only "profit" can be applied as a utility value to mining high-utility itemsets, but "weight", "cost" and other different interesting factors can also construct different types of high-utility itemsets. Several previous works were respectively proposed to reveal the set of complete HUIs in recent years. The concept of high-utility itemset was first introduced by Yao et al. [12]. Then Yao et al. proposed HUIMs to reveal HUIs effectively [13], purchase quantities and unit profits of items are both considered to show high-utility itemsets in a transaction database. Several extensions have been presented and discussed to better provide the efficient data structures and pruning strategies to reduce the search space for revealing HUIs [14–19]. Evolutionary computations algorithms were also applied to find HUIs with limited termination conditions [20–22].

Although HUIM solves the limitations in ARM or FIM, it is based on the concept of minimum utility threshold to determine whether an itemset is a HUI. Moreover, the discovered patterns only reveal the correlation without any quantitative information; it is not a realistic model for many applications. Therefore, the fuzzy-set theory is usually applied to extend the quantitative concept into linguistic terms to resolve this kind of situation. It can provide more flexible and effective knowledge than the previous traditional 0–1 logical concept. Recently, there are more and more machine learning or expert systems applied Fuzzy-set to enhance their performance [23–26]. In this paper, fuzzy-set theory was involved in HUIM by presenting an alternative knowledge for pattern mining. It solves the limitations of the above works especially the quantity is involved as the representative of the linguistic terms that shows interpretable knowledge of the discovered patterns. For example, generic ARM or HUIM cannot present a rule such as "a large number of people that purchase *diapers* implies a few numbers of people purchase *beer*". In many real-world applications and domains, it shows crucial information by involving the quantity factor for pattern representation. Wang et al. [27] first proposed a fuzzy-based high-utility algorithm to incorporate the fuzzy-set concept into HUIM. The fuzzy-set theory first transfers the value of quantities to linguistic terms that can fit the real situations and provide a more detailed pattern in a transaction database. Lan et al. [28] further applied effective upper-bounds to enhance the performance for revealing high fuzzy utility itemsets. Wu et al. [29] introduced an evolutionary computation technique to this field and first proposed a GA-based high fuzzy utility mining algorithm. However, previous works suffer from the "combinational explosion problem" thus the search space to discover the required information is huge. Thus, the above approaches are hard to be applied in a database with massive records and items. Nevertheless, previous algorithms can only be performed on a single machine to handle simple and small databases; they are not easy to handle very large-scale databases.

To solve the above limitations in high fuzzy utility itemset mining, this paper presents an efficient high fuzzy utility itemset mining (EFUPM) algorithm for revealing the high fuzzy utility patterns. To handle big datasets, the Hadoop framework for the proposed EFUPM algorithm is also introduced named HFUPM. The significant contributions of this paper are summarized as follows:

1. We proposed an efficient high fuzzy utility pattern mining algorithm (EFUPM) algorithm to efficiently discover the high fuzzy utility patterns from the database in a single-machine mechanism.
2. To handle the large-scale databases, the Hadoop-based high fuzzy utility pattern mining (HFUPM) algorithm is also introduced here with several MapReduce tasks to scan the original dataset and reduce the size of temporary files as possible. For each MapReduce task (iteration), the developed HFUPM does not perform any modifications or duplicated operators for the original dataset. This improves the feasibility of the developed method in practical applications.
3. The developed HFUPM can also reduce the number of database scans since it could catch all of the information of the currently processed itemset, which is very efficient in the large-scale environment by reducing the seeking time of the database.
4. To better reduce the search space for discovering the high fuzzy utility patterns, two upper-bounds are then designed here to early remove the unpromising candidates that can be applied to both single-machine or Hadoop-based framework for mining the high fuzzy utility patterns.

2. Related work

Frequent itemset (FI) and association rule mining (ARM) [1] are fundamental tasks in data mining. Frequent items are created by a pre-defined threshold from a transaction database for minimum support, and association rules are generated by another predefined threshold that calls for minimum confidence. In the past, ARMs were used in many applications and flourished in true market environments. Apriori [1] is the first transaction dataset mining algorithm to show the FIs and ARMs. Two primary steps are included in Apriori's process to achieve the global. In the first step, the process of Apriori will evaluate all candidate itemset and finds FIs of which the frequencies are larger than the predefined minimum support

threshold. Then the process will generate the new candidate itemsets from the FIs which are identified before. Until no new candidate itemsets are generated, the first step is finished. In the second step, an AR (association rule) generated by FI combination will be revealed if AR confidence is greater than the minimum trust level pre-defined. Apriori is also a "generate-and-test" method that involves multiple scans of the transaction record to produce FIs and reveal ARs. This will cause the awful question of performance to search for a huge dataset. Frequent pattern (FP)-tree, which is a tree structure, was proposed to speed up the discovery of FIs, and the corresponding mining algorithm is called for FP-growth [30]. There are several pieces of research further based on Apriri and FP-growth to enhance the performance for mining FIs and ARs from transaction databases [31–33]. Other works focused on applying these algorithms in the several real-world applications [10,11,4,34]. FI and ARM, nonetheless, have a serious disadvantage. A high-frequency itemset does not necessarily mean it is an actual valuable commodity and has a high value for capital.

To reveal more important information in a dataset high-utility itemset mining (HUIM) was proposed. To find a set of high-utility itemsets (HUIs), it considers the quantity and unit profit of itemsets simultaneously [12,35]. The model TWU (Transaction-weighted utilization) has been designed to reduce the number of HUI revealing items of a candidate because of its DC property, also known as downward closure [13]. When observing the downward closure of transaction-weighted utilization we can say clearly that if a given itemset from which the TWU is greater than some given threshold, we would say that this would be a high TWU itemset, which we can refer to as HTWUIs. Furthermore, we can further this idea by saying that all further supersets are then definitely can not be HTWUIs as well as HUIs. A given method that is used to apply TWU mode is then able to ignore the given itemset of which the superset for the HTWUIs. Li was able to propose what is known as an IIDS, or isolated discarding item strategy that can be used to further lower the count of candidate itemsets through a TWU model [16]. HUI-Miner algorithm depends on a utility-list structure and follows the depth-first search in this structure, it does not need to generate candidates to mine the HUIs [18]. The HUI-Miner uses a vertical database display and is designed with a simple joint operation to avoid multiple database scans, but the high-utility k-itemsets still can be obtained. Several extensions of HUIM (i.e., applied for presenting different knowledge) are widely studied and discussed [36,37].

The previous HUIs mining algorithms hid the quantity information in the output patterns. People could not get the original quantity information from the HUIs; even if HUIs are calculated by the unit profits and quantities. It causes that people might lose some vital knowledge and can not set up precise strategies. Fuzzy theorem, therefore, was involved in the field of HUIM to transfer the quantities to linguistic terms [27]. It shows the semantics concept between the traditional bit logical (0 or 1) in the computer science [38]. Each individual can be assigned a fuzzy value within an interval [0, 1] that indicates the membership level of a certain class. Recently, the fuzzy concept has been widely applied in many applications to provide its flexibility for decision making [39–44]. In this paper, the fuzzy concept extends and converts a traditional item to several fuzzy items. Thus, it enhances the complexity of mining high fuzzy utility itemsets (HFUIs). Lan et al. proposed effective upper-bounds for mining HFUIs and prune the searching space to increase performance [28]. Wu et al. also applied the evolutionary computation technique to this field and first proposed a GA-based high fuzzy utility mining algorithm [29].

In the past, the above algorithms obtained excellent outcomes for mining high-utility itemsets. In a small scale dataset, the previous methods are all good enough to reveal high-utility itemset in reasonable computation time. However, that is because the previous methods applied some complicated operators in datasets. By modifying the original dataset and maintaining a table to store some metadata, the previous algorithms can reduce the times of seeking a database and looking for a transaction. The principal problem of these algorithms is that they are very hard to be applied in a database which includes a huge number of transactions. They modified the original dataset, tried to remove the unnecessary items and transactions, and combined multiple transactions into one transaction. They also maintained a table to stored the metadata for utilities and store in the memory. In the big-data environment, the cost of modifying a dataset is very high and also almost impossible. The storage of the original dataset and the modified dataset is another critical issue. The modifying operators and combining operators would produce several duplications of the original dataset. No matter how to store these data, it lays a big burden for handling a terabyte-scale (or more) dataset and applying these operators. Paradoxically, HUIM is a data mining technic for the commercial field, and most of the real transaction datasets are very large. That is to say, the previous methods are very difficult to be applied in a real-world situation.

Propose by Google in 2008, MapReduce is a popular framework for programming used in big data [45]. It uses a distributed algorithm on clusters and is the combination of two procedures, namely **Map** and **Reduce**. MapReduce uses a key-value list for input data. Each node in the system that is tasked with performing the **Map** is first given this input list. The procedure itself performs a "mapping" which can be customized by the programmer that transfers this input list into another key-value list. Next, a shuffle operator gets defined that distributes records from this new key-value list to the nodes that are tasked to **Reduce**. One of the main goals of the shuffle operator is to maintain balance for each node. Lastly, the **Reduce** procedure goes through a summary operation for all input records which have identical keys and releases the results. Usually, the output as well as a key-value list. The two main procedures run independently and there usually is not any interaction between nodes in a given cluster. Therefore, the qualities of MapReduce usually include reliability and strong distribution to nodes within a given cluster. Dynamically users can add/remove computational units and manage loads on the fly. Moreover, the system can move jobs that have failed from one cluster to another and remotely shutdown nodes that are acting strangely. Overall, MapReduce can provide a dynamic, and reliable framework capable of handling big data.

3. Preliminaries and Problem Statement

Let a quantitative transaction dataset $D = \{T_1, T_2, \cdots, T_n\}$ with n transactions and a finite itemset $I = \{i_1, i_2, \cdots, i_m\}$ with m distinct items. Each transaction $T_q \in D$ includes several items from I and has a unique identifier $q(1 \leqslant q \leqslant m)$ called its *TID*. Moreover, each item $i_j \in T_q$ has a purchase quantity which is denoted as $q(i_j, T_q)$. In order to estimate the utility of each item and transaction, a profit table $P = \{p(i_1), p(i_2), \cdots, p(i_m)\}$ shows the unit profit of each item $i_j (i \leqslant j \leqslant m)$. If an itemset is called as k-itemsets, it means this itemset includes k items and each item belongs to itemsets I. In this study, the same fuzzy membership functions will be applied in each item in I. For example, three linguistic terms indicated *low*, *middle* and *high* are defined in a membership function to estimate the fuzzy utility for each item. Assume a minimum fuzzy utility threshold is set as δ according to users' preference to reveal high fuzzy utility itemsets.

An example for the input data is shown below. A quantitative dataset with purchase quantities is shown in Table 1, which will be running example in this paper. It includes 10 transactions and 6 distinct items, denoted from (a) to (f) respectively. Assume the profit values (external utilities) of each item in Table 1 are shown as $\{a:3, b:9, c:1, d:5, e:6, f:1\}$.

Definition 1 (*Fuzzy set of quantitative value*). The fuzzy set f_{yz} of the purchase value (quantitative value) $q(i_z, T_y)$ of item i_z in the transaction T_y can be denoted by the given membership functions as:

$$f_{yz} = \left(\frac{f_{yz1}}{R_{z1}} + \frac{f_{yz2}}{R_{z2}} + \cdots + \frac{f_{yzh}}{R_{zh}}\right), \tag{1}$$

where h is the number of membership functions (linguistic term or fuzzy region) of an item, f_{yzl} is the fuzzy value of the quantitative value $q(i_z, T_y)$ in the l-th fuzzy region R_{zl}.

Definition 2 (*Item utility of an item*). The utility of an item $i_j$in the transaction $T_q$is denoted as $u(i_j, T_q)$, and is defined as:

$$u(i_j, T_q) = q(i_j, T_q) \times p(i_j). \tag{2}$$

Definition 3 (*Fuzzy utility of a fuzzy item*). The fuzzy utility fu_{yzl} of the l-th fuzzy region for the item i_z in the transaction T_y is defined as:

$$fu_{yzl} = f_{yzl} \times u(i_z, T_y). \tag{3}$$

Note that the membership functions are the same for each item in this study. It is possible to use different membership functions for different items.

Definition 4 (*Fuzzy utility of a fuzzy itemset*). The fuzzy utility fu_{yX} of a fuzzy itemset X in the transaction T_y is the summation of the fuzzy utility of X in T_y. It is defined as:

$$fu_{yX} = \sum_{fi_{zl} \in X} fu_{yzl} \tag{4}$$

The fuzzy item fi_{zl} is the item z with a linguistic term l in the fuzzy itemset X. Note that the definition of the fuzzy utility of a fuzzy itemset is different in the previous works [27,28] but follows the previous GA-based algorithm [29]. In the papers [27,28], the process of calculating the fuzzy utility of a fuzzy itemset is performed by the minimum operator among all fuzzy values in this fuzzy itemset. After that, this single minimum fuzzy value is then used to accumulate the fuzzy utilities of all fuzzy items. However, this model is not suitable for the utility concept because if a certain fuzzy item is with low fuzzy value, the whole fuzzy utility of this itemset is also shallow. It thus will produce incomplete information of the discovered high fuzzy utility patterns.

Table 1
A quantitative database D.

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

Definition 5 (*Actual fuzzy utility of a fuzzy itemset*). The actual fuzzy utility afu_X of a fuzzy itemset X in a transaction database D is the summation of all fuzzy utility of X in all the transaction which includes X. It is denoted as:

$$afu_x = \sum_{I_X \subset y \in D} fu_{yX}, \tag{5}$$

where I_X is the original discrete itemset for X and y is a transaction including I_X.

Definition 6 (*High fuzzy utility itemset*). Set a pre-defined minimum fuzzy utility threshold δ. A fuzzy itemset X is called as a high fuzzy utility itemset, if and only if $afu_X \geqslant \delta$.

Based on the above definitions, the concept of high fuzzy utility pattern mining considers not only the utility of an itemset but also the quantitative information interpreted by fuzzy-set theory. Comparing with the traditional high-utility itemset, a high fuzzy utility pattern provides more useful information to discover complete and understandable information. For example, it can help a manager in the supermarket to decide the purchase volumes of each product.

Problem Statement: Assume a quantitative transaction database D, and each transaction in D is recorded with the purchased items and quantities. A pre-defined high fuzzy utility threshold δ is given. The issue in this paper is to find all the high fuzzy utility patterns in which their actual fuzzy utility values are higher than or equal to the pre-defined threshold. Moreover, to handle the large-scale databases for mining the high fuzzy utility patterns in realistic situations, an efficient MapReduce approach is required to be designed to speed up mining performance.

4. Proposed EFUPM and HFUPM

There are four parts to this section. The first part is that an upper-bound with closure property named as itemset fuzzy upper bound is proposed. The second part is the algorithm for constructing the searching graph to reveal HFUIs. The third one is the pruning strategies proposed in the developed EFUPM. Two upper-bounds names as sub-tree fuzzy utility and local fuzzy utility are introduced and applied in the designed pruning strategy to reduce the search space. Last but not least, the proposed Hadoop-based HFUPM is then implemented and studied.

4.1. Itemset Fuzzy Utility Upper Bound

In traditional FIM, downward closure (DC) property is always applied to limit the searching space for revealing high-frequency itemsets. However, in the field of HUIM, the utility cannot be the assessed value to directly apply DC property. Liu et al. [13] proposed transaction weighted utility *twu* to keep closure property in HUIM. Lan et al. [28] introduced fuzzy utility upper-bound *fuub* to hold the DC property for high fuzzy utility pattern mining. In this paper, a tighter upper-bound named itemset fuzzy utility upper bound *ifuub* is proposed to further reduce the searching space and maintains the DC property. The detailed definitions of the original *fuub* and the developed *ifuub* are described as follows.

Definition 7 (*Maximal fuzzy utility of an item*). The maximal fuzzy utility mfu_{yz} of the item i_z in the transaction T_y is defined as:

$$mfu_{yz} = max\{fu_{yz1}, fu_{yz2}, \cdots, fu_{yzl}\}, \tag{6}$$

where fu_{yzk} is the fuzzy utility of the k-th fuzzy region of the item i_z in the transaction T_y.

The maximal fuzzy utility is the maximal value of an item calculated in a transaction. Because a fuzzy itemset can only select one fuzzy region (linguistic term) of each item, it can be used to generate an upper-bound of fuzzy utility for an itemset with DC property.

Definition 8 (*Maximal transaction fuzzy utility*). The maximal transaction fuzzy utility $mtfu_y$ of the transaction T_y is the summation of the maximal fuzzy utility values of all the items in the T_y, that can be defined as:

$$mtfu_y = \sum_{i_z \in T_y} mfu_{yz}. \tag{7}$$

The maximal transaction fuzzy utility is the maximal fuzzy utility of which an itemset can obtain in a transaction. If an itemset is included in a transaction, the fuzzy utilities of its superset and itself are equal or less than the maximal transaction fuzzy utility. Therefore, it can be used to design an upper-bound and keep the DC property for high fuzzy utility pattern mining.

Definition 9 (*Fuzzy utility upper-bound*). The fuzzy utility upper-bound $fuub_X$ of a fuzzy itemset X is the summation of the maximal transaction fuzzy utility of all the transactions which include I_X (the original discrete itemset of X) in the dataset D. The detailed definition is defined as:

$$fuub_X = \sum_{I_X \subseteq T_y \in D} mtfu_y. \tag{8}$$

In previous work, *fuub* is used to keep the DC property and reduce the search space to reveal high fuzzy utility patterns. Nevertheless, *fuub* is a loose upper-bound for fuzzy utility. Another tighter upper-bound is developed in the proposed algorithm as described below.

Definition 10 (*Itemset maximal transaction fuzzy utility*). The itemset maximal transaction fuzzy utility $imtfu_{yX}$ of the fuzzy itemset X in the transaction T_y is defined as:

$$imtfu_{yX} = fu_{yX} + \sum_{i_z \in T_y, i_z \notin I_X} mfu_{yz}, \tag{9}$$

where I_X is the corresponding discrete itemsets of the fuzzy itemset X. Note that if $I_X \neg \subseteq T_y$, $imtfu_{yX} = 0$, that is because T_y cannot provide fuzzy utility to X. The value of $imtfu_{yX}$ is a maximal fuzzy utility to which the transaction can provide for all of the fuzzy supersets of X.

Definition 11 (*Itemset fuzzy utility upper bound*). The itemset fuzzy utility upper bound $ifuub_X$ of the fuzzy itemset X in the dataset is the summation of the maximal transaction fuzzy utility of all the transactions including I_X (the original discrete itemset of X) in the dataset D. It is then defined as:

$$ifuub_X = \sum_{I_X \subseteq T_y \in D} imtfu_y. \tag{10}$$

Comparing with *mtfu*, *imtfu* applies the fuzzy itemset X itself and not uses the discrete itemset to calculate the upper-bound. Therefore, *imtfu* is equal or less than *mtfu* but also keeps the DC property. The related proofs are shown below.

Theorem 1 (*Anti-monotonicity property of imtfu*). *Assume two fuzzy itemsets X and X' and $X \subset X'$, a transaction T_y such as $I_X \subseteq T$. Thus,*

$$mtfu_y \geqslant imtfu_{yX} \geqslant fu_{yX'}.$$

Proof 1. The proof is divided into three parts.

1. $mtfu_y \geqslant imtfu_{yX}$:

$$\begin{aligned} imtfu_{yX} &= fu_{yX} + \sum_{i_z \in T_y, i_z \notin I_X} mfu_{yz} \\ &\leqslant \sum_{i_z \in I_X} mfu_{yz} + \sum_{i_z \in T_y, i_z \notin I_X} mfu_{yz} \\ &= \sum_{i_z \in T_y} mfu_{yz} \\ &= mtfu_y. \end{aligned}$$

2. $I_{X'} \not\subseteq T$:

$\because fu_{yX'} = 0 \therefore imtfu_{yX} \geqslant fu_{yX'}$.

3. $I_{X'} \subseteq T$:

$$\begin{aligned} [I]imtfu_{yX} &= fu_{yX} + \sum_{i_z \in T_y, i_z \notin I_X} mfu_{yz} \\ &\geqslant fu_{yX} + fu_{y(X' \setminus X)} \\ &\quad + \sum_{i_z \in T_y, i_z \notin I_{X'}} mfu_{yz} \\ &= fu_{yX'} \sum_{i_z \in T_y, i_z \notin I_{X'}} mfu_{yz} \\ &\geqslant fu_{yX'}. \end{aligned}$$

$\because$ 1), 2) and 3) $\therefore mtfu_y \geqslant imtfu_{yX} \geqslant fu_{yX'}$.

Theorem 2 (*Anti-monotonicity property of ifuub*). *Assume two fuzzy itemsets X and X′ and X ⊂ X′, a transaction T_y such as $I_X \subseteq T$. Thus,*

$$fuub_X \geqslant ifuub_X \geqslant afu_{X'}.$$

Proof 2. ∵ Theorem 1 ∴$mtfu_y \geqslant imtfu_{yX} \geqslant fu_{yX'}$ in each transaction T_y where $I_X \in T_y$.

$$\rightarrow \sum_{I_X \subseteq T_y \in D} mtfu_y \geqslant \sum_{I_X \subseteq T_y \in D} imtfu_y \geqslant \sum_{I_X \subset y} fu_{yX'} = \sum_{I'_X \subset y} fu_{yX'} \rightarrow fuub_X \geqslant ifuub_X \geqslant afu_{X'}. \quad \square$$

The designed *ifuub* is a tighter upper-bound than that of the *fuub* and keeps DC property in the mining progress. It can also be applied in an Apriori-based algorithm to reveal high fuzzy utility patterns.

Definition 12 (*k-candidate high fuzzy utility itemset*). An fuzzy itemset *X* is called as *k*-candidate high fuzzy utility itemset *k*-CHFUI if and only if $ifuub_X \geqslant$ minimumfuzzyutilitythresholdδ.

In an Apriori-based algorithm, *k*-CHFUIs can be as candidate fuzzy itemsets with *k* items in each round to reveal the high fuzzy utility itemsets. In the proposed EFUPM and HFUPM algorithms, the 1-CHFUI is used to build the searching graph for the mining process and two upper-bounds are introduced here to further reduce the search space. The detailed descriptions are stated below.

4.2. Searching graph for mining process

4.2.1. Constructing search graph

Before performing the proposed algorithms, a search graph is first built that can be used to apply the pruning strategies and the proposed algorithms. The proposed algorithms discover all the HFUPs (high fuzzy utility patterns) in a dataset when the proposed algorithms estimate or prune all the nodes in a graph. Based on the developed Theorem 2, the search graph

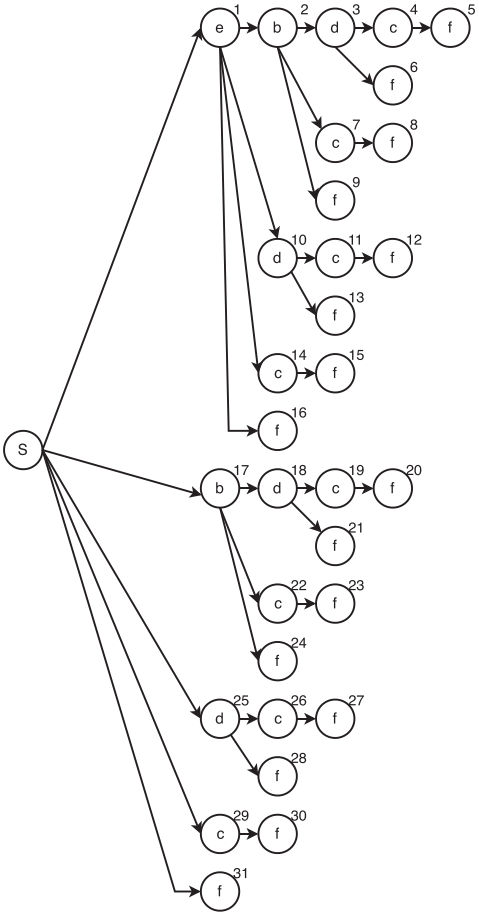

Fig. 1. A search graph with five items.

contains only 1-CHFUIs in a transactional database. After obtaining the 1-CHFUIs, the proposed algorithms sort the corresponding discrete items in the 1-CHFUIs (using the maximal fuzzy value of an item for all fuzzy terms) with their *ifuub* by descending order. After that, a routing graph will be generated from this ordered list. Details are then described in Algorithms 1 and 2 A search graph of the given example is shown in Fig. 1. In this graph, a node (item) represents all of the possible corresponding fuzzy items for this item. For example, if the fuzzy item $\{a.low\}$ and $\{a.middle\}$ exist in 1-CHFUI, a node a represents the fuzzy item $\{a.low\}$ and $\{a.middle\}$ simultaneously.

Algorithm 1. Construct searching graph

Input: an ordered item list l.
Output: a searching graph G.

1: set $G = \varnothing$;
2: put the starting point s into G;
3: run BuildChildNodes(G, s, l);
4: **return** G;

Algorithm 2. Build child nodes algorithm

1: **function** BuildChildNodesG, s, l
2: **for** each item i in l **do** ▷ select item i by l order.
3: $G \leftarrow i$;
4: generate a directed link from s to i;
5: **if** item i is not the last one in list l **then**
6: set a list l_i is a sub list of l after item s;
7: run BuildChildNodes(G, i, l_i);
8: **end if**
9: **end if**
10: **end function**

4.2.2. The Node's Expression for Itemset

According to the above process, a search graph is generated before EFUPM. Therefore, each node in the search graph indicates a specific fuzzy itemset which can be estimated whether it is a high fuzzy utility pattern. The expression of a node is the traveling log between the starting node to this node. For example, assume there are three linguistic terms as *low*, *middle*, and *high*. If a tracking history of the search process is *e* and *b*, and $e.low$, $b.middle$ and $b.high$ exist in 1-CHFUI, the corresponding fuzzy item is $\{e.low, b.middle\}$, and $\{e.low, b.high\}$. However, fewer possible fuzzy itemsets will be estimated due to the developed pruning strategies as follows.

4.3. Pruning strategies

For the developed EFUPM and HFUPM, the pruning strategies used in EFIM [46] are then utilized and enhanced here to early prune the unpromising candidates, thus the search space can be greatly reduced. Two upper-bounds named *sub-tree fuzzy utility* and *local fuzzy utility* are then designed here to determine whether the process is required to estimate the following nodes from the current node. Details are then given below.

Definition 13 (*Sub-tree fuzzy utility*). Assume the item order is $I = \{i_1, i_2, \ldots, i_k\}$, and the last item in a sorted fuzzy itemset X sorted by I order is fi_m. The fi_o is defined as the first fuzzy item following fi_m and assume the corresponding discrete item is set as i_o. I' is a sub-list of I, which is defined as $\{i_{o+1}, i_{o+2}, \ldots, i_k\}$. Assume that $z = i_n$ is an item $\in I'$ and $g(X \cup \{z\})$ is a transaction set in which transaction including itemset $X \cup \{z\}$. The *sub-tree fuzzy utility* of z w.r.t X is:

$$sfu(X, z) = \sum_{T \in g(X \cup (z))} [fu(X, T) + fu(z, T) + \sum_{i \in T \cap I'} mfu(i, T)]. \tag{11}$$

Definition 14 (*Local fuzzy utility*). Assume the item order is $I = \{i_1, i_2, \ldots, i_k\}$. Also assume the last item in a sorted fuzzy itemset X sorted by I order is fi_m. fi_o is the first fuzzy item following fi_m. The corresponding discrete item is i_o. Let I' be a sub-list of I as $\{i_{o+1}, i_{o+2}, \ldots, i_k\}, z = i_n$ is an item $\in I'$ and $g(X \cup \{z\})$ is a transaction set in which transaction including itemset $X \cup \{z\}$. The *local fuzzy utility* of z w.r.t X is:

$$lfu(X,z) = \sum_{T \in g(X \cup (z))} \left[fu(X,T) + \sum_{i \in T \cap l'} mfu(i,T) \right]. \quad (12)$$

The *sub-tree fuzzy utility*, *sfu* and *local fuzzy utility*, *lfu* are respectively corresponding to the *sub-tree utility* and *local utility* that were used in EFIM model [46]. They are the tighter upper-bounds for the high fuzzy utility patterns in the proposed search graph. Thus, if the *sfu* value of a fuzzy itemset is less than the threshold, the following fuzzy itemsets in the search graph are no possibility to be a high fuzzy itemset. Moreover, if the *lfu* value of a fuzzy itemset is less than the threshold, all of the super fuzzy itemsets of this fuzzy itemset are also no possibility to be a high fuzzy itemset. The developed algorithm is then provided below.

4.4. Developed EFUPM algorithm

The proposed EFUPM applies the developed high fuzzy utility version (*sub-tree fuzzy utility* and *local fuzzy utility*) to establish the upper-bounds of candidates in the search graph. The detailed pseudo-code is provide following. The proposed EFUPM follows the process of the previous EFIM algorithm that applies sub-tree (fuzzy) utility and local (fuzzy) utility for performing the pruning strategies that can be used to greatly reduce the size of search space. To mainly focus on the developed Hadoop-based algorithm (HFUPM) in this paper, detailed algorithm can be referred to [46]. However, due to involvement in the fuzzy theorem, the proposed process of EFUPM is more complicated than the previous EFIM. Note that, the following description also matches the concept of the following HFUPM Hadoop framework. The proposed algorithm first calculates the value of *ifuub* for each item in the input dataset. According to the pre-defined membership function set, there are several corresponding fuzzy items of an item. Therefore, the item list l will be sorted by the maximal value of *ifuub* for each item by descending order. This list will be used to establish a searching graph logically. It is different from the traditional EFIM algorithm, each node in this searching graph indicates a set of fuzzy item. For example, if a node is a, it might indicate the fuzzy items *a.low*, *a.middle* and *a.high*. Then, the proposed will search the graph by the breadth-first order. The searching process will start from a node in the searching graph and estimate the following nodes directly linked to this node. For example, it might be as $\{a.low, b.high\} \cup \{c\}$. Then, the proposed process will calculate all of the values of *fu* for all possible fuzzy itemsets and check it is HFUIs or not. The pruning method will be performed consequently. *sfu* is calculated to check the proposed process should establish the following node or not. And *lfu* is calculated to maintain to set r_l. This set can be used to check a fuzzy itemset should be evaluated or not. In the proposed HFUPM framework, the pruning method will be handle by a task file and be described in the following session.

4.5. Developed HFUPM algorithm

The framework of the proposed HFUPM is shown in Fig. 2. It consists of three MapReduce architectures, which are responsible for different purposes in the proposed framework. The first one is used to reveal the 1-CHFUIs, the second one is to sort the 1-CHFUIs and the last one is for the mining task of high fuzzy utility patterns. An independent process between MapReduce 2 and 3 is to generate a task file for discovering the high fuzzy utility patterns in MapReduce 3. Each part is respectively described below.

4.5.1. Transaction dataset

Generally, a Hadoop program catches the input data from HDFS with the plain text format. In the developed HFUPM, the transaction information is stored in HBase with the binary structure. HFUPM tries to produce less temporary data as possible, therefore, the random access ability for the original input dataset is very important. Apache HBase provides the random access ability and good performance to reach any transaction in the dataset. For any specific transaction, HFUPM uses a hash table to store the transaction information and uses Apache Avro to serialize it into the binary structure. Thus, HFUPM can obtain specific information for a certain item in a transaction efficiently. All the complex data structures are serialized and de-serialized by Apache Avro in the developed HFUPM. The tasks of each MapReduce is then described below.

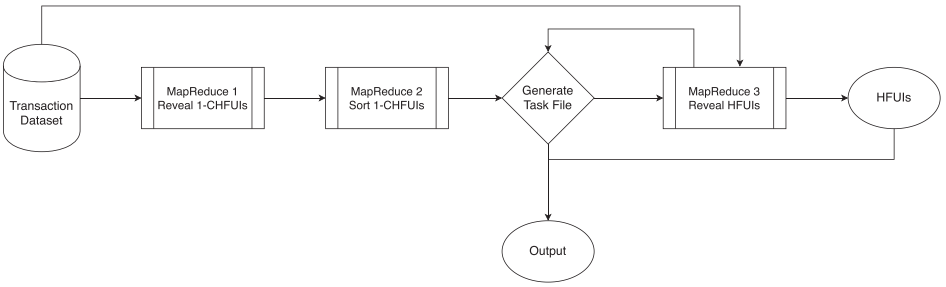

Fig. 2. Proposed HFUPM framework.

4.5.2. MapReduce 1: Revealing 1-CHFUIs

MapReduce 1 is set at the beginning of the proposed framework that reveals all 1-CHFUIs in the dataset. Besides Mapper and Reducer, HFUPM also builds the Combiner in MapReduce 1 to increase the performance. Detailed descriptions are shown as follows.

In the first MapReduce framework, each Mapper obtains a part of transaction dataset. Each item in a transaction will be transferred to several fuzzy items based on the pre-defined membership functions. The key-value pair for the fuzzy item and the *imtfu* of this fuzzy item will be considered as the output to the later Combiners. In the Combiner operator, the Mapper nodes accumulate the value of the same item ID before it outputs the key-value pair list to Reducers. Actually, the output value of the node is the partial *ifuub* of an item. It can reduce the requirement of the communication bandwidth and the time of transformation. Finally, the key-value pairs of the same item ID will be assigned to the same Reducer. The Reducers in MapReduce 1 then calculates the sum of the partial *ifuub* of each item and outputs the 1-CHFUIs to MapReduce 2 for the sorting process.

4.5.3. MapReduce 2: Sorting 1-CHFUIs

MapReduce 2 is an option in the proposed framework. That is because the scale of the number of items always not achieves at the big data level. Sorting thousands of 1-CHFUIs in a single machine is always faster than a Hadoop cluster. However, if a dataset contains a huge number of 1-CHFUIs, MapReduce 2 is designed to handle this situation to increase the performance. MapReduce 2 is applied to a famous MapReduce sorting algorithm named TeraSort for sorting 1-CHFUIs. It uses the auto-sorting method for keys in the MapReduce process, whereas the default sorting action in Hadoop is sorting keys in ascending order. Note that users need to overwrite the compare function to change the sorting behavior. The algorithm of MapReduce 2 is very simple, it just switches keys and values to sort by *ifuub*.

In the default setting, a Reducer sorts keys for its own data automatically. That is to say, Hadoop can not guarantee global sorting. It is not a sorted list by *ifuub* descending order. Thus, users need to overwrite the shuffle function defined in the Hadoop framework. The shuffle function is used to assign a record to a specific Reducer. In the developed HFUPM, assume there are n Reducers in the system, the shuffle function needs to sample n - 1 *ifuub* values and sorts them by the descending order. This sorting sample list will be the cutting points for global sorting. The records of which *ifuub* is larger than or equal to the first value in the sorting sample list will be assigned to the first Reducer, the records of which *ifuub* is less than the last value in the list will be assigned to the last Reducer and the records of which *ifuub* is less than the $(n$ - $1)$-th value and larger than or equal to n-th value will be assigned to the n-th Reducer. In general, the sorting process can be performed in a single machine without the Hadoop framework.

Algorithm 3. Generate the task file for candidate k-HUIs

Input: a list l of key-value pair for the previous task file. The *sub-tree fuzzy utilities* and local fuzzy utilities for the previous candidate fuzzy itemsets.
Output: a list o of key-value pair for the current candidate itemsets.

```
1: for each (n, l_f) in l do
2:   set nextNodes = {}, wholeFollowingList = {};
3:   for each n_f in l_f do
4:     if sfu(n, n_f) ⩾threshold then                ▷ sfu is sub-tree fuzzy utility
5:       nextNodes ← n_f;
6:     end if
7:     if lfu(n, n_f) ⩾threshold then                ▷ lfu is local fuzzy utility
8:       wholeFollowingList ← n_f;
9:     end if
10:  end for
11:  if nextNodes is not {} then
12:    for each nn in nextNodes do
13:      nodeFollowing = {};
14:      index = wholeFollowingList.getIndex(nn) + 1;
15:      length = wholeFollowingList.length;
16:      while index is not equal to length do
17:        temp = wholeFollowingList.get(index);
18:        nodeFollowing ← temp;
19:        index = index + 1;
20:      end while
21:      if nodeFollowing is not {} then
22:        o ← ({n, m}, nodeFollowing);
23:      end if
```

24: **end for**
25: **end if**
26: **end for**
27: **Output** o to the task file;

4.5.4. Generate task file

In MapReduce 3, while seeking the dataset each time, the proposed HFUPM will reveal all of the high fuzzy utility patterns (HFUPs) containing the same number of items. Thus, a task file is required to be generated to show the candidate itemsets for MapReduce 3. In other words, at the k-th time by performing MapReduce 3, all of the k-HFUPs will be discovered in this round. The format of the task file is a list of the key-value structure, and the key is an itemset which is estimated at the previous round, and the value is a list of items which can be extended from this itemset in the search graph. Therefore, the keys and the values can be combined with the new candidate itemset for the iteration. The first task file thus contains one record of which key is *NULL* and value is the list of 1-CHFUIs. The description of generating task files at other rounds is shown in Algorithm 3.

In Algorithm 3, it combines Algorithm 1 and the developed pruning strategies to reduce the search space of the unpromising candidates. In the designed Algorithm, two emptysets are created to store the fuzzy itemsets whose *sfu* or *luf* (Line 2, Algorithm 3) is larger than the pre-defined threshold. The *nextNodes* is a set where each node has possibility to be extended for generating more high fuzzy utility itemsets. A loop is then performed to traverse all nodes in *nextNodes* in Lines 12 to 24. On the other hand, any branch following a node should exist in *wholeFollowingList*; otherwise, it will not seek the possible high fuzzy utility itemsets. In Lines 13 to 20, the proposed algorithm seeks the branches whether it exists in *wholeFollowingList*. After that, the algorithm generates a task file with the current node if the related *nodeFollowing* is not empty (Lines 21 to 23, Algorithm 3). An example is shown below to explain the process clearly. Assume the current node n is 17 ({b}), *nextNodes* = {d} and *wholeFollowingList* = {d, c}. Due to *sub-tree utility* pruning strategy, the following branches after node 22 and 24 will be pruned. The branch between node 18 and 21 will also be pruned, because $lu(b,f) <$ threshold. Finally, $\{\{b,d\},\{c\}\}$ will be written into list o in this loop iteration. After performing Algorithm 3 for the other nodes with 2-items, all of the candidates with 3-items will be put into the output list o. Note that, the value of *sfu* and *lfu* are the maximal value of all the fuzzy terms, and the corresponding candidates of high fuzzy utility patterns will be put in the task file. For example, if $sfu(\{b.low\},d.high) <$ *threshold* but $sfu(\{b.low\},d.middle) \geqslant$ *threshold*, we will simply indicate $sfu\{b,d\} \geqslant$ *threshold* and keep the node d in the search graph. However, the fuzzy itemset $\{b.low,d.high\}$ will not be existing in the task file.

4.5.5. MapReduce 3: Revealing HFUPs

In this section, the developed HFUPM loads the potential HFUPs from the task file and transaction information from the HBase database. Then, a MapReduce 3 will be utilized to calculate the *fuzzy utility*, *sub-tree fuzzy utility* and *local fuzzy utility* for all candidates, and further reveals the HFUPs in this dataset. In MapReduce 3, HFUPM also maintains a related transaction ID list (this list records the ID numbers and does not include any transaction information) to be as an input file for the MapReduce system. Therefore, the HFUPM does not need to scan the whole dataset, and the computational cost can be greatly reduced.

Algorithm 4. Mapper of MapReduce 3

Input: a list l of transaction ID, a task file f and a transaction dataset d.
Output: a list o of key-value pair. The key is an itemset and the value is the utility information for a certain transaction which contains this itemset.

1: **for** each transaction ID i in l **do**
2: set *nextIteration* = false; 3: **for** each (n, l_f) in f **do**
4: get transaction t from d using ID i
5: **if** t does not contain n **then**
6: continue;
7: **end if**
8: calculate $u(n,t)$;
9: set *futilities* = {}, *sfUtilities* = {}, *lfUtility* = 0, *existFollowing* = false;
10: **for** each n_f in l_f **do**
11: **if** t contains n_f **then**
12: *existFollowing* = true; ▷ It exists candidate itemsets.

(*continued on next page*)

```
13:          calculate u(n_f, t);
14:          lfUtility = lfUtility + u(n_f, t);
15:          put (n_f → u(n_f, t)) into futilities;
16:          put (n_f → 0)) into sfUtilities;                    ▷ Initialization.
17:          for each (k → v) in sfUtilities do
18:            put (k → v + u(n_f, t)) into sfUtilities;         ▷ Update.
19:          end for
20:        end if
21:      end for
22:      if existFollowing is false then
23:        continue;
24:      end if
25:      nextIteration  = true;                                  ▷ Keep this transaction.
26:      for each (k → v) in utilities do
27:        set temp = {}, temp2 = {};
28:        temp ← (n, k);
29:        temp2 ←{(futility: u(n, t) + v), (sfUtility: u(n, t)+ sfUtilities.get(k)), (lfUtility:
   u(n, t) + lUtility)};
30:        o ← (temp, temp2);
31:      end for
32:    end for
33:    if nextIteration is true then
34:    put i into the new transaction ID list;Save in the HBase's table for the next iteration.
35: end if
36: end for
```

In Algorithm 4, a Mapper calculates the *fuzzy utility*, *sub-tree fuzzy utility*, and *local fuzzy utility* (Line 29) of each itemset in the task file for each transaction which is assigned to this Mapper. If a transaction contains a candidate itemset, it will be kept in the new transaction ID list; otherwise, it will be removed. For the same reason as MapReduce 1, MapReduce 3 also makes a Combiner to reduce the communication cost. The description of the Combiner in MapReduce 3 is shown below.

Algorithm 5. Reducer of MapReduce 3

Input: a list l of key-value pair. The key is an itemset and the value is the partial utility information for this itemset.
Output: a list o of key-value pair. The key is an itemset and the value is the utility information for this itemset. And a part of HUIs to HBase.

```
1: for each key-value pair (i, l_ui) in l do
2:    set fut = 0, sfUt = 0, lfUt = 0;
3:    for each ui in l_ui do
4:      fut = fut + ui.get(futility);
5:      sfUt = sfUt + ui.get(sfUtility);
6:      lfUt = lfUt + ui.get(lfUtility);
7:    end for
8:    set temp = {};
9:    temp←{(futility: fut), (sfUtility: sfUt), (lfUtility: lfUt)};
10:   o ← (i, temp);
11:   if fut ⩾ threshold then
12:     HUIs ← i;                                  ▷ It is stored in the HBase database.
13:   end if
14: end for
```

In Algorithm 5, HFUPM calculates the value of *fuzzy utility* (Line 4), *sub-tree fuzzy utility* (Line 5) and *local fuzzy utility* (Line 6) of each candidate for the whole transaction dataset and outputs to the HBase database in order to be used for the next iteration. If the *fuzzy utility* of a fuzzy itemset is larger than the pre-defined threshold, this itemset will be stored in the HBase database (Line 12). Finally, if Algorithm 3 cannot generate more candidates for the HFUPs, it indicates that the developed HFUPM has already revealed all of the HFUPs in this transaction dataset; the process is then terminated and the results are then output as the discovered HFUPs.

5. Experimental evaluation

This section examines the effectiveness of the EFUPM, HEUPM, and the previous Apriori-based algorithms for high fuzzy utility pattern mining under different databases. The experiments were conducted in computing nodes equipped with the Intel Core i7-6700 CPU @ 3.40 GHz *8 and 32 GB assigned RAM, running Linux Ubuntu 16.04 LTS. To achieve parallelization, the experiment was run under the Hadoop2.5.1 cluster with one master node and five data nodes.

5.1. Data information and preparation

The original databases include many transactions with items and their utilities. To facilitate calculation, firstly, the experiment initializes the database and take the line number as TID. Each transaction contains three parts: the first part is the items, the second part is the utility of each item, and the third part is the *ifuub*of the transaction. By analyzing these data sets, corresponding high fuzzy utility patterns can be mined. To analyze the different performance between the three algorithms more comprehensively and show the performance differences between different databases, six standard datasets for evaluating the traditional HUIM algorithms are selected as input data (namely BMS, Mushroom, Accident, Connect, Foodmart, and Chess), and they are shown in Table 2. Besides, a pre-defined membership function is also provided to calculate the fuzzy value of each item. Moreover, to verify the advantages and disadvantages of the algorithm in big data, the experiments are divided into three groups, namely small, medium, and large of the conducted datasets. After preparing these databases, we need to upload them to HDFS for the initialization for performing the Hadoop-based HFUPM algorithm.

5.2. Algorithm development and evaluation

The proposed algorithms can be adjusted according to requirements. Before performing the algorithms, a threshold value is required to be set for determining whether an itemset is a high fuzzy utility pattern. In the designed HFUPM, MapReduce2 is an iterative process, and every iteration needs to scan the database to calculate the fuzzy utilities of the itemsets. When the database is extensive, it often takes a long time to scan the database. Therefore, the performance of EFUPM is related to the number of items in the databases. The first experiment aims to compare the running time of the four algorithms in six different datasets. The information of the six databases is shown in Table 2. The four algorithms are the proposed EFUPM, HFUPM, and the traditional Apriori and the Hadoop-based Apriori framework (Apriori(M)) [47]. Note that, to handle the process for mining HFUPs, the proposed *ifuub* concept will be applied in the Apriori-based algorithms.

5.2.1. Runtime

As is shown in Fig. 3, by taking the first group of databases which has smaller size, the runtimes of EFUPM and Apriori is significantly less than that of HFUPM and Apriori(M). In this figure, The *x*-coordinate represents the different databases within the group, and the *y*-coordinate represents the runtime of the compared algorithms. From the experimental results, it can be seen that the MapReduce framework of Hadoop has no advantage while dealing with small datasets. This is reasonable since it takes a certain amount of time to startup the Hadoop cluster, and communication cost among different nodes is also very expensive. When we use the first group of databases (small size), for the HFUPM and Apriori(M) algorithms, the size of the four databases is not up to the size of a block. Therefore, the process is done on one node. Even so, the two algorithms will start the whole Hadoop cluster using the master node scheduling the entire process, but the EFUPM and Apriori complete in a single machine. Therefore, while using this set of data, the performance of single-machine algorithms is better than Hadoop-based frameworks. Moreover, the performance difference between HFUPM and Apriori(M) at low data volumes is not significant because the amount of computation is small.

The next experiments take the second group of databases which has a medium size, and the results are shown in Fig. 4. From the results, it can be seen that the runtime of the single machine algorithms is still less than Hadoop-based frameworks. In this experiment, the size of the input data exceeded the size of the block, and the parallel computing was implemented in the Hadoop-based algorithms; the input data was divided into different nodes for calculation. This step saves a lot of computational costs. However, as mentioned earlier, while using Hadoop's MapReduce architecture, the communication cost among different nodes is very high. Moreover, since EFUPM and Apriori algorithms run on a single machine, its performance requirements on the computer begin to increase as the size of the dataset gets larger. And as the amount of compu-

Table 2
Database information.

Dataset	Trans.	Items	Avg. trans. length
BMS	59,601	497	4.8
Mushroom	8,124	119	23.0
Accident	340,183	468	33.8
Connect	67,557	129	43.0
Chess	3,196	75	37.0
Foodmart	4,141	1,559	4.4

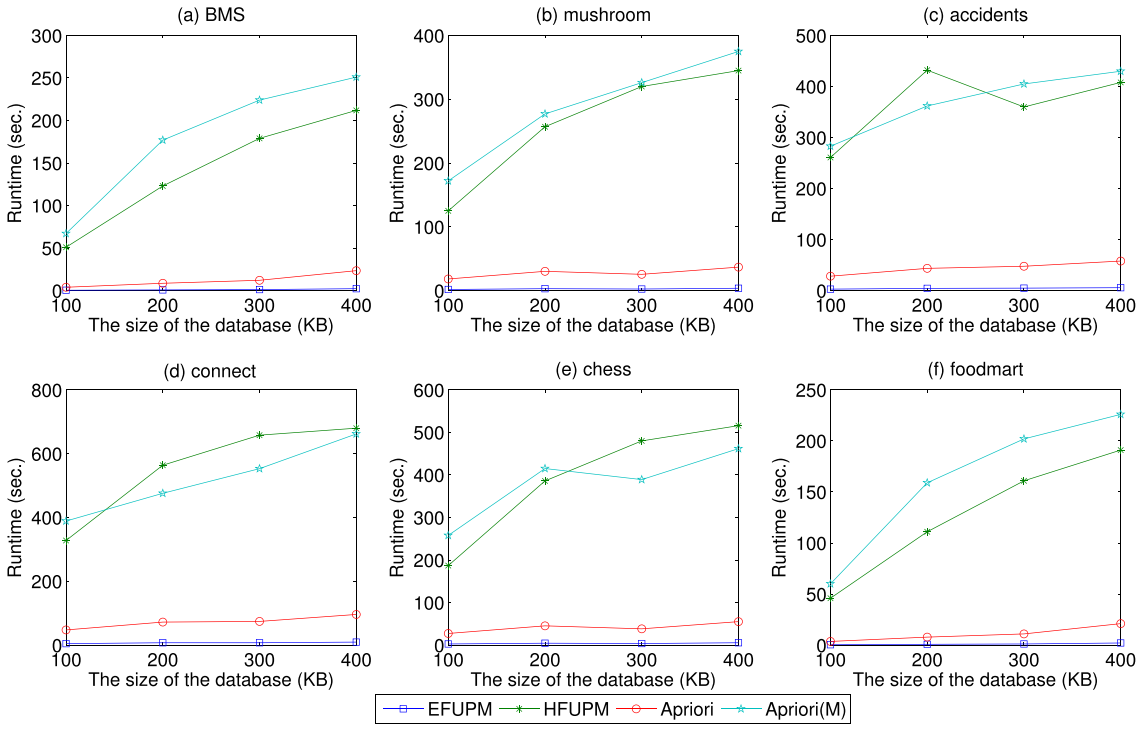

Fig. 3. Runtimes in the small size databases.

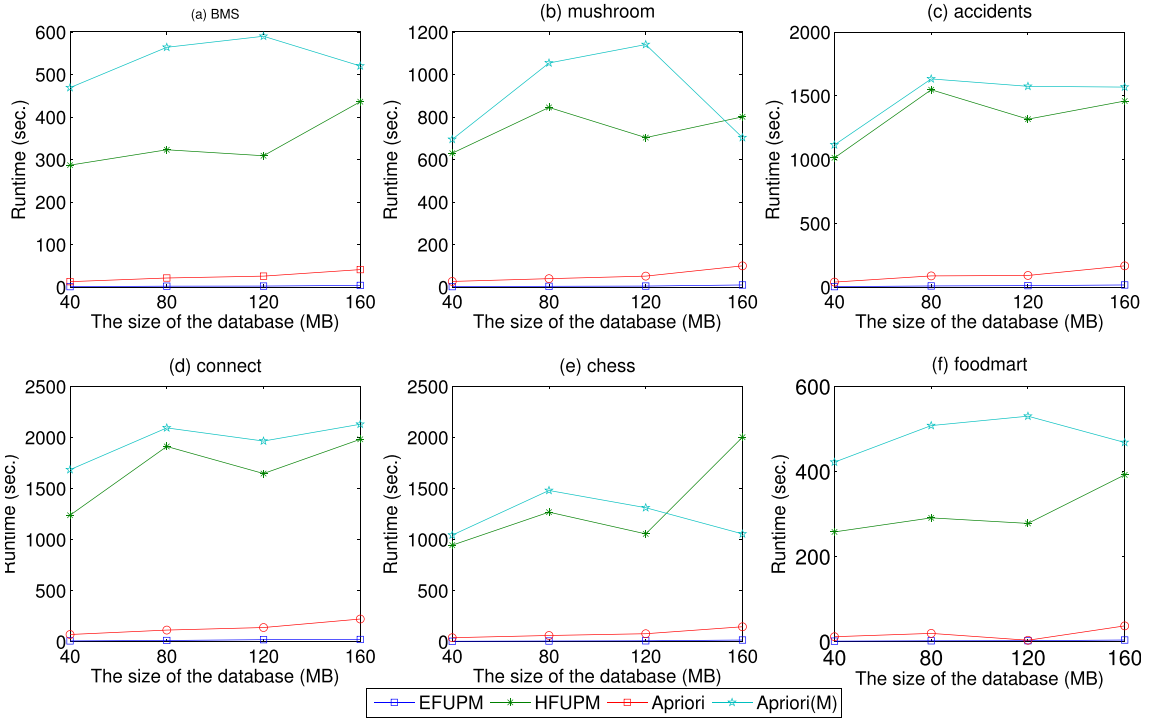

Fig. 4. Runtimes in the medium size databases.

tation increased, the performance difference between HFUPM and Apriori(M) is obvious. The reason is that the developed upper-bounds are efficient to reduce the search space; it can save much time in the calculation process.

In the third experiment, the input data is the large datasets, and the results of this experiment are shown in Fig. 5. The results of this experiment showed the significance of our developed HFUPM. Obviously, in this experiment, the running time of both Hadoop-based algorithms has increased a lot, but the runtime of the Apriori(M) algorithm grows faster. What is more, when the size of the database reaches 1.5 GB, the single machine can no longer support to perform the EFUPM and Apriori algorithms. For the HFUPM and Apriori(M) algorithms, although they take a long time to initialize the MapReduce framework, the final results can still be obtained, and we can see more clearly that the HFUPM performs better than the Apriori(M).

In summary, the experiments showed that the HFUPM can be well performed for mining the high fuzzy utility patterns in a very large database. At the same time, the purpose of using the MapReduce architecture in Hadoop is usually not to achieve parallelism and improve algorithm performance but to process big data that cannot be processed by a single machine. Also, the performance of the single-machine EFUPM algorithm is better than the Apriori-based algorithm for mining the high fuzzy utility patterns.

5.2.2. Memory usage

According to the previous works [1], the Apriori(M) generates a lot of intermediate data during computation, which takes up more storage space. The designed HFUPM can thus solve this limitation. In these experiments, we compared the file sizes generated by HFUPM and Apriori(M) while they were performed and executed. The different databases and memory usage are independent, thus the influence of memory usage depends on the size of a database. Therefore, we only present the relationship between the size of a database and the memory usage in the experiments. The experimental results are shown in Table 3.

To minimize the impact of different factors on the performance of the compared algorithms, the number of items remains stable while adjusting the database size. However, the transaction size is then changed. From the results, it can be seen that the Apriori(M) has a better memory usage while performing it in a tiny database. That is because the proposed HFUPM needs to perform a very complex process in the search space to reduce the unpromising candidates. However, while the size of the database increases, HFUPM shows excellent memory usage compared to the Apriori(M); it kept the size of memory in a suitable rage, but Apriori(M) suffered the memory explosion crisis.

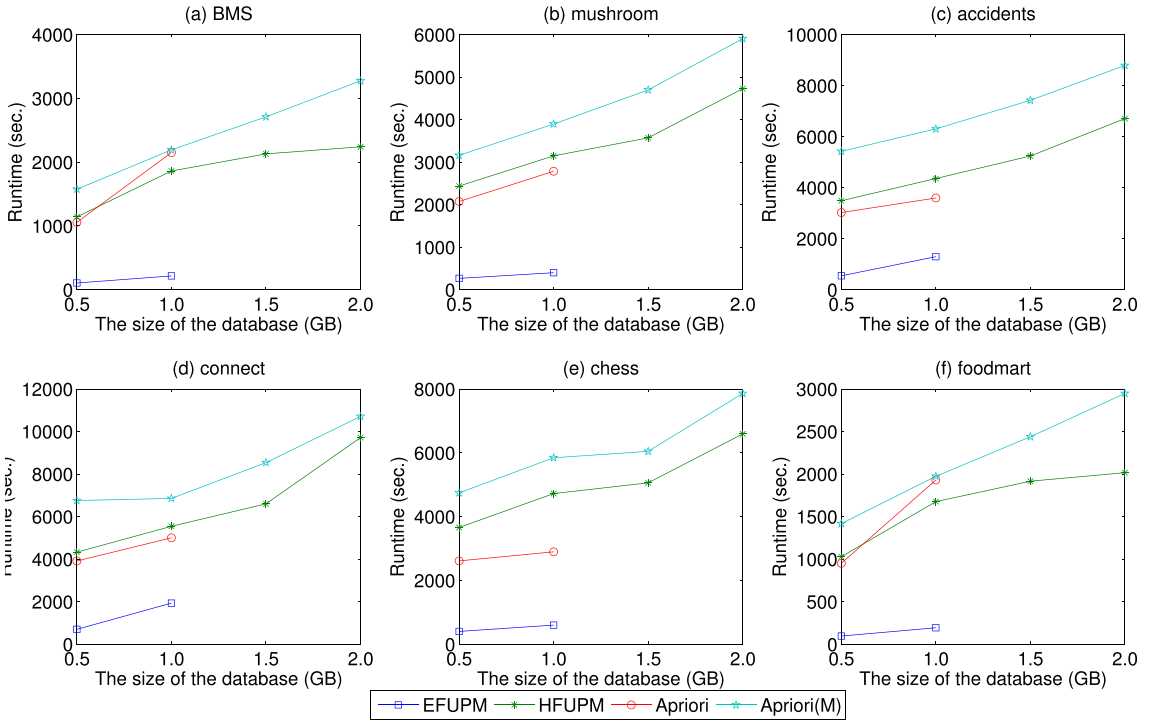

Fig. 5. Runtimes in the large size databases.

Table 3
Memory usage of two compared algorithms in three databases.

	Database size		
Algorithm	100 KB	40 MB	0.5 GB
HFUPM	9 MB	10 MB	17 MB
Apriori(M)	4 MB	97 MB	190 MB

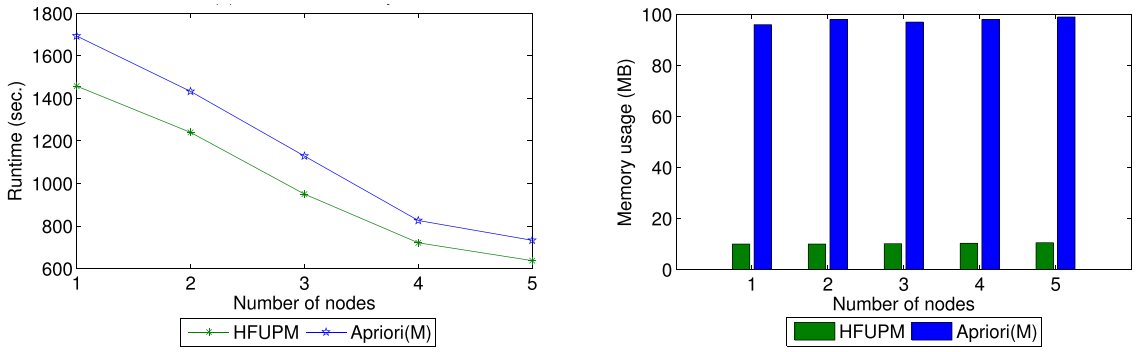

Fig. 6. Scalability comparisons.

5.2.3. Scalability of the Hadoop-based algorithms

In this section, the influence of varied numbers of nodes for the developed HFUPM and the Apriori(M) is shown below. The runtimes and memory usages with 1 to 5 computers in the Hadoop cluster are respectively shown in Fig. 6(a) and (b). Here, to make a fair comparison, only the algorithm based on the MapReduce technique is considered in the experiments. For different datasets, the influences between the varied numbers of nodes and the runtime are very similar. Therefore, we simply showed the relationship between the scalability results regarding the number of nodes and runtime in one database. The experimental results were performed in the "accident" database with 46M size.

From the conducted results, it is obvious to see that the developed HFUPM always has better performance than the Apriori(M) under a varied number of nodes in the Hadoop cluster. The reason is that the developed upper-bounds can provide better pruning strategies to reduce the search space of the unpromising candidates and the developed HFUPM has a better ability to manage memory, thus the developed HFUPM can better reveal the high fuzzy utility patterns (HFUPs) in a parallel environment and does not need too much memory usage for the progress. Moreover, these two frameworks are both suitable to be utilized in the MapReduce system. When the number of nodes increases, the reduction of the runtime is very obvious. It can be concluded that these two algorithms can be parallelly performed in a distribution system. On the other hand, the memory usages are almost the same while the number of nodes increases for the two compared algorithms. Thus, we can say that more nodes in the Hadoop system can enhance the performance for pattern mining based on the developed frameworks.

6. Conclusion

With the rapid growth of commercial datasets, it is necessary to develop an effective mining algorithm to reveal knowledge for decision making. However, the traditional mining algorithms were limited in the ability to handle the huge dataset. In this paper, we designed two algorithms named EFUPM and HFUPM for mining high fuzzy utility patterns. Two new upper-bounds are also estimated and developed in this paper to reduce the search space of the unpromising candidates. Moreover, the HFUPM is implemented by the Hadoop framework to handle the large-scale databases. Experiments showed that the developed EFUPM has better performance than that of the Apriori-like approach for mining the high fuzzy utility patterns and the designed HFUPM has good performance than the Apriori-based model running on the Hadoop framework. Thus, the designed algorithms provide a practical model in real-life situations.

In this paper, the designed HFUPM is based on the Hadoop architecture for implementation. In the future, the Spark framework is then applied to the designed model for handling the high fuzzy utility patterns mining. More constraints such as incremental or multi-objective concepts can also be considered to contribute to the significant benefits in the field of pattern mining. The machine learning model can also be involved to predict the high fuzzy utility patterns in the sequential databases as the future study.

CRediT authorship contribution statement

Jimmy Ming-Tai Wu: Conceptualization, Methodology. **Gautam Srivastava:** Writing - review & editing. **Min Wei:** Formal analysis, Validation. **Unil Yun:** Formal analysis, Validation. **Jerry Chun-Wei Lin:** Conceptualization, Methodology.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: International Conference on Very Large Data Bases, vol. 1215, 1994, pp. 487–499..
[2] R. Agrawal, R. Srikant, Mining sequential patterns, in: International Conference on Data Engineering, 1995, pp. 3–14..
[3] P. Fournier-Viger, J.C.W. Lin, R.U. Kiran, Y.S. Koh, R. Thomas, A survey of sequential pattern mining, Data Science and Pattern Recognition 1 (1) (2017) 54–77.
[4] R. Martinez, N. Pasquier, C. Pasquier, Genminer: mining non-redundant association rules from integrated gene expression data and annotations, Bioinformatics 24 (22) (2008) 2643–2644.
[5] W. Pedrycz, M. Krawczak, S. Zadrozny, Computational intelligence techniques for decision support, data mining and information searching, Information Sciences 460–461 (2018) 374–376.
[6] J.C.W. Lin, Y. Shao, Y. Djenouri, U. Yun, Asrnn: A recurrent neural network with an attention model for sequence labeling, Knowledge-Based Systems (2020).
[7] J.C.W. Lin, G. Srivastava, Y. Zhang, Y. Djenouri, M. Aloqaily, Privacy preserving multi-objective sanitization model in 6g iot environments, IEEE Internet of Things Journal (2020).
[8] M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 866–883.
[9] C.H. Yun, M.S. Chen, Using pattern-join and purchase-combination for mining web transaction patterns in an electronic commerce environment, International Computer Software and Applications Conference (2000) 99–104.
[10] C. Creighton, S. Hanash, Mining gene expression databases for association rules, Bioinformatics 19 (1) (2003) 79–86.
[11] E. Georgii, L. Richter, U. Rückert, S. Kramer, Analyzing microarray data using quantitative association rules, Bioinformatics 21 (suppl_2) (2005) ii123–ii129..
[12] H. Yao, H.J. Hamilton, C.J. Butz, A foundational approach to mining itemset utilities from databases, in: International Conference on Data Mining, 2004, pp. 482–486.
[13] Y. Liu, W. k. Liao, A. Choudhary, A fast high utility itemsets mining algorithm, in: International Workshop on Utility-based Data Mining, 2005, pp. 90–99..
[14] A. Erwin, R.P. Gopalan, N. Achuthan, Efficient mining of high utility itemsets from large datasets, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2008, pp. 554–561.
[15] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, Y.K. Lee, Efficient tree structures for high utility pattern mining in incremental databases, IEEE Transactions on Knowledge and Data Engineering 21 (12) (2009) 1708–1721.
[16] Y.C. Li, J.S. Yeh, C.C. Chang, Isolated items discarding strategy for discovering high utility itemsets, Data & Knowledge Engineering 64 (1) (2008) 198–217.
[17] B.E. Shie, V.S. Tseng, P.S. Yu, Online mining of temporal maximal utility itemsets from data streams, in: ACM Symposium on Applied Computing, 2010, pp. 1622–1626..
[18] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in: ACM International Conference on Information and knowledge Management, 2012, pp. 55–64.
[19] J. Liu, K. Wang, B.C. Fung, Mining high utility patterns in one phase without generating candidates, IEEE Transactions on Knowledge and Data Engineering 28 (5) (2015) 1245–1257.
[20] S. Kannimuthu, K. Premalatha, Discovery of high utility itemsets using genetic algorithm with ranked mutation, Applied Artificial Intelligence 28 (4) (2014) 337–359.
[21] J.C.W. Lin, L. Yang, P. Fournier-Viger, T.P. Hong, M. Voznak, A binary pso approach to mine high-utility itemsets, Soft Computing 21 (17) (2017) 5103–5121.
[22] J.M.T. Wu, J. Zhan, J.C.W. Lin, An aco-based approach to mine high-utility itemsets, Knowledge-Based Systems 116 (2017) 102–113.
[23] X. Huo, L. Ma, X. Zhao, B. Niu, G. Zong, Observer-based adaptive fuzzy tracking control of mimo switched nonlinear systems preceded by unknown backlash-like hysteresis, Information Sciences 490 (2019) 369–386.
[24] Q. Wu, L. Zhou, Y. Chen, H. Chen, An integrated approach to green supplier selection based on the interval type-2 fuzzy best-worst and extended vikor methods, Information Sciences 502 (2019) 394–417.
[25] J. Zhou, Q. Zhang, X. Li, Fuzzy factorization machine, Information Sciences 546 (2020) 1135–1147.
[26] K. Zhang, J. Zhan, X. Wang, Topsis-waa method based on a covering-based fuzzy rough set: an application to rating problem, Information Sciences 539 (2020) 397–421.
[27] C.M. Wang, S.H. Chen, Y.F. Huang, A fuzzy approach for mining high utility quantitative itemsets, in: International Conference on Fuzzy Systems, 2009, pp. 1909–1913.
[28] G.C. Lan, T.P. Hong, Y.H. Lin, S.L. Wang, Fuzzy utility mining with upper-bound measure, Applied Soft Computing 30 (2015) 767–777.
[29] J.M.T. Wu, J.C.W. Lin, P. Fournier-Viger, T. Wiktorski, T.P. Hong, M. Pirouz, A ga-based framework for mining high fuzzy utility itemsets, in: International Conference on Big Data, 2019, pp. 2708–2715.
[30] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, Data Mining and Knowledge Discovery 8 (1) (2004) 53–87.
[31] Y.C. Li, C.C. Chang, A new fp-tree algorithm for mining frequent itemsets, in: Advanced Workshop on Content Computing, 2004, pp. 266–277..
[32] G. Grahne, J. Zhu, Fast algorithms for frequent itemset mining using fp-trees, IEEE Transactions on Knowledge and Data Engineering 17 (10) (2005) 1347–1362.
[33] C. Lucchese, S. Orlando, R. Perego, Fast and memory efficient mining of frequent closed itemsets, IEEE Transactions on Knowledge and Data Engineering 18 (1) (2005) 21–36.
[34] M.S. Chen, J.S. Park, P.S. Yu, Efficient data mining for path traversal patterns, IEEE Transactions on Knowledge and Data Engineering 10 (2) (1998) 209–221.
[35] H. Yao, H.J. Hamilton, L. Geng, A unified framework for utility-based measures for mining itemsets, in: ACM SIGKDD Workshop on Utility-Based Data Mining, 2006, pp. 28–37.

[36] W. Gan, J.C.W. Lin, J. Zheng, H.C. Chao, H. Fujita, P.S. Yu, Proum: Projection-based utility mining on sequence data, Information Sciences 513 (2020) 222–240.
[37] H. Nam, U. Yun, E. Yoon, J.C.W. Lin, Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions, Information Sciences 529 (2020) 1–27.
[38] L.A. Zadeh, Fuzzy sets, Information and Control 8 (3) (1965) 338–353.
[39] F. Kutlu Gündoğdu, C. Kahraman, Spherical fuzzy sets and spherical fuzzy topsis method, Journal of Intelligent & Fuzzy Systems 36 (1) (2019) 337–352.
[40] F.K. Gündoğdu, C. Kahraman, A novel fuzzy topsis method using emerging interval-valued spherical fuzzy sets, Engineering Applications of Artificial Intelligence 85 (2019) 307–323.
[41] P. Peykani, E. Mohammadi, A. Emrouznejad, M.S. Pishvaee, M. Rostamy-Malkhalifeh, Fuzzy data envelopment analysis: an adjustable approach, Expert Systems with Applications 136 (2019) 439–452.
[42] I. Škrjanc, J.A. Iglesias, A. Sanchis, D. Leite, E. Lughofer, F. Gomide, Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: a survey, Information Sciences 490 (2019) 344–368.
[43] Y. Pan, L. Zhang, Z. Li, L. Ding, Improved fuzzy bayesian network-based risk analysis with interval-valued fuzzy sets and ds evidence theory, IEEE Transactions on Fuzzy Systems (2019).
[44] L. Fei, Y. Deng, Multi-criteria decision making in pythagorean fuzzy environment, Applied Intelligence 50 (2) (2020) 537–561.
[45] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications 51 (1) (2008) 107–113.
[46] S. Zida, P. Fournier-Viger, J.C.W. Lin, C.W. Wu, V.S. Tseng, Efim: a fast and memory efficient algorithm for high-utility itemset mining, Knowledge and Information Systems 51 (2) (2017) 595–625.
[47] Y.C. Lin, C.W. Wu, V.S. Tseng, Mining high utility itemsets in big data, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2015, pp. 649–661.