Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

vol. 15, no. 10, October 2019, pp. 2805-2816
DOI: 10.23940/ijpe.19.10.p27.28052816

# A Distributed Frequent Itemset Mining Algorithm for Uncertain Data

Jiaman Ding[a,b,*], Haibin Li[a,b], Yang Yang[a,b], Lianyin Jia[a,b], and Jinguo You[a,b]

[a]*Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, 650500, China*
[b]*Artificial Intelligence Key Laboratory of Yunnan Province, Kunming, 650500, China*

**Abstract**

With the rapidly expansion of big data in all domains, it has become a major research topic to improve the performance of mining frequent patterns in massive uncertain datasets in recent years. Most conventional frequent pattern mining approaches take expect, probability, or weight as one single factor of item support, and algorithms that consider both probability and weight are unable to balance execution efficiency under the circumstances of big data. Therefore, we propose a distributed frequent itemset mining algorithm for uncertain data: Dfimud. Firstly, Dfimud calculates the maximum probability weight value of 1-items and prunes the items whose value is less than the given threshold. Secondly, to reduce the times of scanning the datasets, a distributed Dfimud-tree structure inspired by FP-Tree is designed to mine frequent patterns. Finally, experiments on publicly available UCI datasets demonstrate that Dfimud achieves more optimal results than other related approaches across various metrics. In addition, the empirical study also shows that Dfimud has good scalability.

*Keywords*: data mining; uncertain data; frequent itemset; distributed framework

(Submitted on September 15, 2019; Revised on September 30, 2019; Accepted on October 25, 2019)



## 1. Introduction

Frequent itemset mining (FIM) is one of the core tasks in the field of data mining. Traditional frequent pattern mining is mostly based on static databases [1-3]; however, most of the data in the real world is uncertain. Frequent pattern mining [4] based on uncertain data has been extensively studied. Traditional algorithms are usually classified into three categories: some based on the expected support model, some based on the probabilistic support model, and others based on the weighted frequent patterns mining model.

First, the expected support model patterns mining algorithms consider that an itemset is an expected frequent pattern if its expected support in an uncertain database exceeds a user-specified minimum support threshold. Chui et al. first proposed the Uapriori algorithm [5], which mines expected support model patterns in uncertain datasets using a level-wise approach based on the well-known Apriori algorithm [6]. The minimum expected support threshold measure was defined to mine frequent patterns in uncertain databases. However, for the candidate set generation, it is still costly, especially in large databases. To address this issue, Wang et al. designed the MBP algorithm [7] to mine uncertain frequent patterns efficiently by reducing the generation of candidate sets. Sun et al. then presented the IMBP algorithm [8] based on an MBP approach, using a novel strategy to discover frequent patterns. Experiments showed that the IMBP algorithm outperforms the MBP approach for mining frequent patterns in uncertain datasets. Leung et al. proposed the UF-growth algorithm [9] based on the FP-Growth algorithm. The algorithm uses a UF-tree to store frequent 1-itemsets and obtains all frequent patterns through a recursive process or by a combination method. The FP-Growth-based algorithm was shown to be much faster than the algorithms inspired by the Apriori algorithm for discovering uncertain frequent patterns. However, the items are merged together in the tree only when they are same and have same existential probabilities in transactions. The UF-growth algorithm causes a large amount of computational time and memory consumption because it generates too many nodes in the tree. To address this issue, Lin et al. proposed the tree-based CUFP-growth algorithm to mine frequent patterns from a compressed uncertain frequent pattern CUFP-tree structure [10]. In the CUFP-tree, the items are the same but have different

* Corresponding author.
*E-mail address*: tjoman@126.com

expected counts in transactions that can be merged together, so the tree size is smaller when compared to the UF-tree. At the same time, this algorithm has better time and space complexity. All the above algorithms belong to the expected support model pattern mining algorithm.

Bernecker et al. introduced a new probabilistic formulation [11] of frequent patterns based on possible world semantics to solve the probability-based model pattern mining problem. However, this formulation ignores the critical problem of generating association rules on probabilistic databases. Thus, Sun et al. proposed two novel algorithms [12], which found frequent patterns in bottom-up and top-down manners; it also explained how to generate association rules.

All of the mentioned algorithms only consider the absolute or relative frequency of items to mining patterns, and they ignore the importance or interest of the itemsets when only the support value of itemsets is calculated. Thus, those that are not frequent but have high importance itemsets may generally be abandoned during the mining process. In order to address this issue, some algorithms that consider the weight support have been proposed. The weighted frequent itemset mining (WFIM) algorithm [13] considers the weight constraint when mining patterns by using a pattern-growth approach. WFIM exploits the weight limits and the minimum weight strategies to keep the downward closure property. Yun et al. extended the method of using weights in pattern mining to find weighted sequential patterns. They proposed an efficient algorithm [14] called WSpan to mine weighted sequential patterns. WSpan discovers weighted sequential patterns by applying the maximal weight among all items as the maximum weight of each sequence. To further enhance the performance of finding weight frequent patterns, Lan et al. proposed a new weight upper-bound model [15] to reduce unpromising candidates in the mining process. In addition, the mining weight support frequent pattern algorithm from uncertain databases [16-17] has been recently studied. However, none of these works consider both the probability and the weight of items. Jerry Chun et al. thus proposed a new pattern called high expected weighted itemset (HEWI) [18]. They designed the HEWI-Uapriori algorithm, but it increases the time complexity of the algorithm.

It is difficult to handle big data on a single computer. To improve the efficiency of data processing, many researchers use distributed computing frameworks to store and process data. Among them, Spark is currently the most widely used distributed computing framework. One of Spark's most important features is memory-based computing, which stores intermediate results in memory rather than on a disk. It greatly reduces unnecessary I/O consumption. Qiu et al. proposed an algorithm called YAFIM [19], which is a parallel Apriori algorithm based on the Spark. However, the YAFIM algorithm is only a parallelized version of the Apriori algorithm, and it still has the problem of generating too many candidate sets and accessing the database many times. Hence, Rathee et al. extended YAFIM and proposed a new algorithm [20] called R-Apriori, which dramatically simplifies computational complexity by eliminating the candidate generation step and reducing costly comparisons. Kumar et al. then introduced a novel Spark-based algorithm called HFIM [21], which solved the problem of scanning complete databases too many times by exploiting a vertical layout manner. Experimental results showed that HFIM outperforms other Spark-based algorithms. However, mining frequent patterns in uncertain databases in a distributed way has not been proposed yet.

In this paper, a novel frequent pattern tree, the distributed frequent itemset mining algorithm for uncertain data tree (Dfimud-tree), is designed. Each node in Dfimud-tree stores an item and its $maxwp(x, t_n)$. A distributed frequent itemset mining algorithm for uncertain data (Dfimud) is then proposed to discovered frequent patterns. Because both the weight and the probability are considered in frequent patterns, more meaningful and useful information can be discovered.

The rest of this paper is organized as follows. Preliminaries are reviewed in Section 2. The Dfimud algorithm and an example are presented in Section 3. Experimental results that demonstrate the performance of the proposed algorithms are provided in Section 4. Conclusions are finally given in Section 5.

## 2. Preliminaries

Let $I = \{i_1, i_2, \cdots, i_m\}$ be a finite set of $m$ distinct items. An uncertain database is a set of transactions $T = \{t_1, t_2, \cdots, t_n\}$, where each transaction $t_q \in T$ is a subset of $I$ and has a unique identifier, called its *ID*. A probability $p(i_m, t_n)$ is appointed to a item $i_m$ in a transaction $t_n$.

Moreover, $W = \{w(i_1), w(i_2), \cdots, w(i_m)\}$ is a weight table, where $w(i_m)$ is the weight of an item $i_m$.

**Definition 1** $w(x, t_n)$ is the weight of an itemset $x$ in a transaction and is defined as Equation (1).

$$w(x, t_n) = \begin{cases} \frac{\sum_{i_m \in x} w(i_m, t_n)}{|k|}, & x \text{ is multiple sets} \\ w(i_m), & x \text{ is single item sets} \end{cases} \quad (1)$$

Where $|k|$ is the number of items in $x$.

**Definition 2** $p(x, t_n)$ is the probability of an itemset $x$ in a transaction $t_n$ and is defined as Equation (2).

$$p(x, t_n) = \begin{cases} \prod_{i_m \in x} p(i_m, t_n), & x \text{ is multiple sets} \\ p(i_m, t_n), & x \text{ is single item sets} \end{cases} \quad (2)$$

**Definition 3** $expsup(x, t_n)$ is the expected support of an itemset $x$ in an uncertain dataset $D$ and is defined as the sum of the expected probabilities of $x$ in the transactions containing $x$, as shown in Equation (3).

$$expsup(x, t_n) = \sum_{x \subseteq t_n \cap t_n \subseteq T} p(x, t_n) \quad (3)$$

**Definition 4** $ewsup(x)$ is the expected weighted support of an itemset $x$ in an uncertain dataset $D$ and is defined as the weight of $x$ multiplied by the expected support of $x$, as shown in Equation (4).

$$ewsup(x) = w(x, t_n) \times expsup(x, t_n) \quad (4)$$

## 3. The Dfimud Method

In this section, we propose a novel Dfimud algorithm to discover the frequent patterns in uncertain datasets. The algorithm consists of two main process: (1) construct the Dfimud-tree and (2) discover the frequent patterns. Now, we can define our procedure.

**Definition 5** $maxw(t_n)$ is the transaction max weight of a transaction $t_n$ and is defined as Equation (5).

$$maxw(t_n) = max\{w(i_1, t_n), \cdots, w(i_m, t_n)\} \quad (5)$$

**Definition 6** $maxp(t_n)$ is the transaction max probability of a transaction $t_n$ and is defined as Equation (6).

$$maxp(t_n) = max\{p(i_1, t_n), \cdots, p(i_m, t_n)\} \quad (6)$$

**Definition 7** $maxwp(t_n)$ is the transaction max weighted probability of a transaction $t_n$ and is defined as Equation (7).

$$maxwp(t_n) = maxw(t_n) \times maxp(t_n) \quad (7)$$

**Definition 8** $maxwp(x)$ is the itemset max weighted probability of an itemset $x$ and is defined as Equation (8).

$$maxwp(x) = \sum_{x \subseteq t_n \wedge t_n \subseteq T} maxwp(t_n) \quad (8)$$

Where $x$ is an itemset, $t_n$ is a transaction whose $ID$ number is $n$, and $T$ is an uncertain database.

**Definition 9** An itemset $x$ in $T$ is called a high weighted probability itemset if its $maxwp(x)$ is greater than or equal to the minimum support, where the minimum support is equal to the minimum support threshold $\varepsilon$ multiplied by the number of transactions in $T$, that is, $maxwp(x) \geq \varepsilon \times |T|$.

**Definition 10** An itemset $x$ in $T$ is a frequent pattern if its $ewsup(x)$ is no less than the minimum support, that is, $ewsup(x) \geq \varepsilon \times |T|$.

**Theorem 1** Let $x^k$ be a $k$-itemset and $x^{k-1}$ be a length $k$-1 subset of $x^k$. If $x$ is a high weighted probability itemset, any subset of $x$ is also a high weighted probability itemset. Thus, $maxwp(x^k) \leq maxwp(x^{k-1})$.

$$maxwp(x^k) = \sum_{x^k \subseteq t_m \wedge t_m \subseteq T} maxwp(t_n) \leq \sum_{x^{k-1} \subseteq t_m \wedge t_m \subseteq T} maxwp(t_n) = maxwp(x^{k-1})$$

Therefore, if $x^k$ is a high weighted probability itemset, any subset of $x$ is also a high weighted probability itemset.

**Theorem 2** The probability of any itemset $x$ in a transaction $t_n$ is always no greater than $maxp(t_n)$.

$$p(x, t_n) \leq maxp(t_n)$$

**Theorem 3** The weight of any itemset $x$ in a transaction $t_n$ is always no greater than $maxw(t_n)$.

$$w(x, t_n) \leq maxw(t_n)$$

**Theorem 4** If an itemset $x$ is not a high weighted probability itemset in an uncertain database, any superset of $x$ is also not a frequent pattern.

$$ewsup(x) \leq maxwp(x)$$

**Proof:**

(1) $p(x, t_n) \leq maxp(t_n)$, $maxp(t_n) = \max\{p(i_1, t_n), \cdots, p(i_m, t_n)\}$

When $x$ is multiple sets:

$$p(i_m, t_n) \in (0,1] \Rightarrow p(x, t_n) = \prod_{i_m \in x} p(i_m, t_n) \leq p(\text{i\_m, t\_n}) \leq maxp(t_n)$$

When $x$ is single item sets:

$$p(x, t_n) = p(i_m, t_n) \leq maxp(t_n)$$

(2) $w(x, t_n) \leq maxw(t_n)$, $maxw(t_n) = \max\{w(i_1, t_n), \cdots, w(i_m, t_n)\}$

When $x$ is multiple sets:

$$w(x, t_n) = \frac{\sum_{i_m \in x} w(i_m, t_n)}{|k|} \leq \frac{\max\{w(i_m, t_n)\} \times |k|}{|k|} = maxw(t_n)$$

When $x$ is single item sets:

$$w(x, t_n) = w(i_m) \leq maxw(t_n)$$

(3) $ewsup(x) \leq maxwp(x)$

$$ewsup(x) = w(x, t_n) \times \sum_{x \subseteq t_m \wedge t_m \subseteq T} p(x, t_n) = \sum_{x \subseteq t_m \wedge t_m \subseteq T} w(x, t_n) \times p(x, t_n)$$
$$\leq \sum_{x \subseteq t_m \wedge t_m \subseteq T} maxw(t_n) \times maxp(x, t_n) = \sum_{x \subseteq t_m \wedge t_m \subseteq T} maxp(t_n) = maxwp(x)$$

*3.1. The Dfimud-Tree Construction*

We propose a Dfimud-tree that uses items $maxwp(x, t_n)$ to finds all high weighted probability itemsets. Then, it calculates

the *ewsup*($x$) of each high weighted probability itemset and filters out the frequent patterns by comparing it with the minimum support.

We store the original dataset file in a Hadoop distributed file system (*HDFS*). At the beginning of the algorithm, the transactional datasets are loaded into *RDD0*:<*trans*>, and the number of partitions are set, where trans refers to transactions.

Then, we transform *RDD0*:<*trans*> to *RDD1*:<*ID*, (*key*, *p*, *w*)> by using the *map*() function, where *ID* is a unique identifier of transaction, *key* is the item, and *p* and *w* represents the probability and weight of an item, respectively. The minimum support threshold is defined as $\varepsilon \in (0,1]$. The *map*() and *flatMap*() functions are applied to *RDD1* to produce *RDD2*: <*ID*, (*key*, *p*, *w*, *maxwp*($t_n$))>. The completed *RDD2* is grouped by using the *groupByKey*() function, generating *RDD3*:<*ID*, (*key*, *p*, *w*, *maxwp*($x$))>. Then, unpromising candidate sets are pruned by using the *filter*() function. Items in *RDD3* are collected into drivers to produce *I-list*, and then *I-list* is sorted in descending order of the *maxwp*($x$) value. Items in *I-list* are grouped to generate *G-list*, and each group has a unique identifier *GID*. *RDD0* is scanned again, and the transactions are updated in the order of items in *I-list*. Then, a new *RDD4:* <*GID*, (*key*,*maxwp*($x$))> is generated. Notice that each item existing in *RDD0* also exists in *G-list*. The *Repartition*() function map items that have same *GID* to the same worker and produce *RDD5*: <*GID*, (*key*, *maxwp*($x$))>. The pseudo-code is depicted in Algorithm 1. Then, a Dfimud-tree is built in each worker, and all high weighted probability itemsets are found and stored in *RDD6*:<*key*, *maxwp*($x$)>.

The proposed Dfimud-tree is a variant of the FP-tree. Each node in the Dfimud-tree stores an item and *maxwp*($t_n$) of transaction in which the item is located. When the Dfimud-tree is constructed, our proposed Dfimud algorithm recursively mines frequent patterns from this tree in a manner similar to FP-growth except for the following:

(1) The Dfimud algorithm utilizes Dfimud-tree (instead of FP-tree) for mining.

(2) When a new pattern x is inserted into the Dfimud-Tree, we store its *maxwp*($x$) to the node at the same time except its occurrence.

All high weighted probability itemsets that satisfy $maxwp(x) \geq \varepsilon \times |T|$ are found. The pseudo-code is depicted in Algorithm 2. The flowchart of this phase is shown in Figure 1.

**Algorithm 1** The Generation Algorithm of RDD5

Phase1 Algorithm

```
//Input: Database
//Output: RDD5
1.RDD0←Database
2.RRD1←Random RDD.p.w
3.RDD2←calculate trans.max.wp
4.RDD3←calculate.item.maxwp
5.I-List←sortBy(RDD3.item.maxwp>
RDD3.item.maxwp)
6.G-List←groupByKey(I-List)
7.If RDD0.item∈G-List
    RDD4←RDD0.item
8.RDD5←repartition(RDD4)
9.Return RDD5
```

**Algorithm 2** Process of Constructing Dfimud-Tree

Dfimud-tree Algorithm

```
//Input: RDD5
//Output: RDD6
1.Root←null
2.For each trans in RDD5
      For each item in trans
          If item in tree
              tree.node.maxwp=tree.node.maxwp
              +RDD5.item.maxwp
          Else create new tree.node
              tree.node.maxwp=item.node.maxwp
      Endfor
  Endfor
3.While(tree.node.number!=1)
      conditionalFP-tree
      mine conditionalFP-tree
4.Return RDD6
```

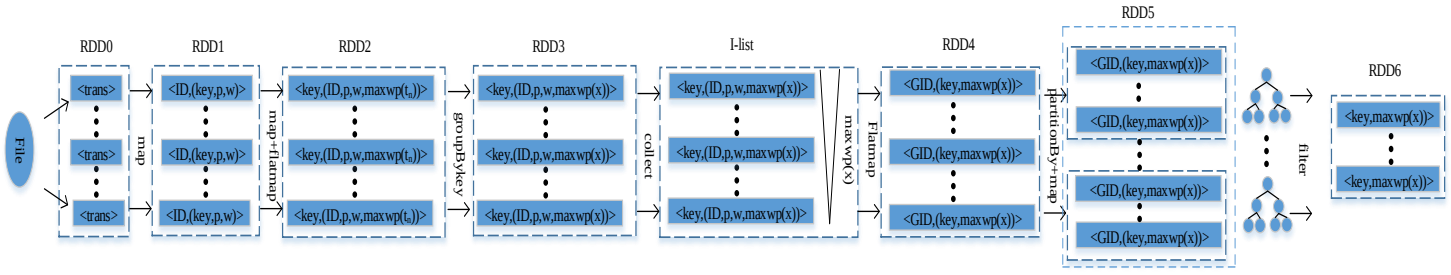



Figure 1. Phase 1 of Dfimud

### 3.2. The Mining Algorithm

To mine the frequent patterns, a second filter is required. To do that, we calculate *ewsup*($x$) of each high weighted probability itemset and compare it with $\varepsilon \times |T|$, define the item as a frequent pattern if its *ewsup*($x$) is greater than or equal to the minimum support $\varepsilon \times |T|$, and then generate *RDD8*:<*key*, *ewsup*($x$)>. The pseudo-code is depicted in Algorithm 3. The flowchart of this phase is shown in Figure 2.

**Algorithm 3** Mining Frequent Pattern Method

Phase3 Algorithm

```
//Input: RDD6
//Output: RDD8
1.RDD7←calculate RDD6.item.ewsup
2.If RDD7.item.ewsup>=minsup
      RDD8←item
3.Return RDD8
```

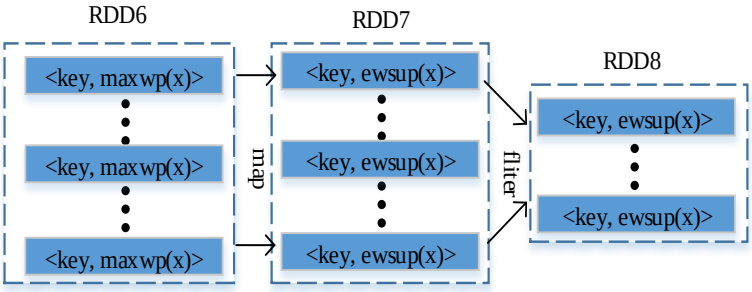



Figure 2. Phase 2 of Dfimud

### 3.3. An Example of Dfimud

The proposed Dfimud takes the input as the database shown in Table 1. For a user-specified minimum support threshold $\varepsilon$, we assume that $\varepsilon = 0.3$, and $|T| = 10$ in this example. Hence, minsupport = $\varepsilon \times |T| = 0.3 \times 10 = 3$.

Table 1. Original database

| Transactions |
|---|
| L N X Y |
| X Z |
| L N Y |
| X Y |
| M N Y |
| M N |
| N X Z |
| M N |
| X Y Z |
| L M N |

The database is extended to an uncertain dataset, as shown in Table 2. According to Equation (5), $maxw(t_1) = \max\{w(L, T_1)$ and $w(N, t_1)$, $w(X, t_1)$, $w(Y, t_1)\} = \max\{0.5, 1.0, 0.3, 0.8\} = 1.0$. According to Equation (6), $maxp(t_1) = \max\{p(L, t_1)$, $p(N, t_1)$, $p(X, t_1)$ and $p(Y, t_1)\} = \max\{0.5, 0.25, 0.4, 0.2\} = 0.5$. According to Equation (7), $maxwp(t_1) = maxw(t_1) \times maxp(t_1) = 1.0 \times 0.5 = 0.5$. Moreover, the $maxw(t_n)$, $maxp(t_n)$, $maxwp(t_n)$ of each transaction is shown in Table 3.

Table 2. Uncertain datasets

| (*ID*, *key*, *p*, *w*) | (*ID*, *key*, *p*, *w*) | (*ID*, *key*, *p*, *w*) |
|---|---|---|
| (1,L,0.5,0.5) | (4,X,0.7,0.3) | (7,Z,0.7,0.6) |
| (1,N,0.25,1.0) | (4,Y,0.8,0.8) | (8,M,0.6,0.9) |
| (1,X,0.4,0.3) | (5,M,0.8,0.9) | (8,N,0.8,1.0) |
| (1,Y,0.2,0.8) | (5,N,1.0,1.0) | (9,X,0.4,0.3) |
| (2,X,0.5,0.3) | (5,Y,0.2,0.8) | (9,Y,0.8,0.8) |
| (2,Z,1.0,0.6) | (6,M,0.8,0.9) | (9,Z,1.0,0.6) |
| (3,L,1.0,0.5) | (6,N,1.0,1.0)) | (10,L,1.0,0.5) |
| (3,N,0.2,1.0) | (7,N,0.8,1.0) | (10,M,0.7,0.9) |
| (3,Y,0.8,0.8) | (7,X,0.6,0.3) | (10,N,0.4,1.0) |

According to Equation (8), $L$ appears in transactions $t_1$, $t_3$, and $t_{10}$. Thus, $maxwp(L) = maxwp(t_1) + maxwp(t_3) + maxwp(t_{10}) = 0.5 + 1.0 + 1.0 = 2.5 < \varepsilon \times |T| = 0.3 \times 10 = 3$. As a result, itemset($L$) is not a high weighted probability itemset. The final calculated high weighted probability itemsets are {$maxwp(M) = 3.8$, $maxwp(N) = 6.1$, $maxwp(X) = 3.34$, $maxwp(Y) = 3.94$}. As shown in Table 4, items are sorted by $maxwp(x)$ descending order.

In this paper, we assume that Spark divides items into two groups as shown in Table 5, group 0 is map by {NY}, and group 1 is map by {MX}. Then, a Dfimud-tree is built in each group to find all high weighted probability itemsets, as described below.

Table 3. *Maxwp*(*tn*) of transactions

| (*ID*,*maxw*(*tn*),*maxp*(*tn*),*maxwp*(*tn*)) |
|---|
| (1,1.0,0.5,0.5) |
| (2,0.6,1.0,0.6) |
| (3,1.0,1.0,1.0) |
| (4,0.8,0.8,0.64) |
| (5,1.0,1.0,1.0) |
| (6,1.0,1.0,1.0) |
| (7,1.0,0.8,0.8) |
| (8,1.0,0.8,0.8) |
| (9,0.8,1.0,0.8) |
| (10,1.0,1.0,1.0) |

Table 4. New database

| (*ID*,item) |
|---|
| (1,N Y X) |
| (2,X) |
| (3,N Y) |
| (4,Y X) |
| (5,N Y M) |
| (6,N M) |
| (7,N X) |
| (8,N M) |
| (9,Y X) |
| (10,N M) |

Table 5. Database after grouped

| Group 0(NY) | Group 1 (MX) |
|---|---|
| (1,N Y) | (1,N Y X) |
| (2, ) | (2,X) |
| (3,N Y) | (3,) |
| (4,Y) | (4,Y X) |
| (5,N Y) | (5,N Y M) |
| (6,N) | (6,N M) |
| (7,N) | (7,N X) |
| (8,N) | (8,N M) |
| (9,Y) | (9,Y X) |
| (10,N) | (10,N M) |

(1) The root of Dfimud-tree is created and labelled with "Null".

(2) The scan of the first transaction leads to the construction of the first branch of the tree: {(N: 0.5), (Y: 0.5)}. Notice that the items in the transaction are listed according to the order in the list of Table 4, and the support of each node is the $maxwp(t_n)$ of this transaction. $maxwp(t_1) = 0.5$; so the support of this transaction is 0.5.

(3) There are no items in the second transaction; thus, scan the third transaction, since its item list {N, Y} shares a common prefix {N, Y}. With the existing path {N, Y}, the support of each node along the prefix is incremented by the value of $maxwp(t_3)$.

(4) The scan of the fourth transaction leads to the construction of the second branch of the tree, {(Y, 0.64}.

(5) Based on this method, the Dfimud-tree in group 0 is constructed as shown in Figure 3.

Construct the Dfimud-tree in group 1, as shown in Figure 4. Once Dfimud-trees are constructed, our proposed Dfimud

algorithm finds high weighted probability itemsets from these Dfimud-trees.

Below is an example of a high weighted probability itemset from the Dfimud-tree in group 1 with minsupport = $\varepsilon \times |T|$ = 0.3×10 = 3. It starts with item X, with *maxwp*(X) = 3.34. Dfimud extracts from three paths-names: (1) ⟨(N, 0.5),(Y, 0.5)⟩ occurs with (X, 0.5), (2) ⟨(Y, 1.44)⟩ occurs with (X, 1.44), and (3) ⟨(N, 0.8)⟩ occurs with (X, 0.8) and forms the {X}-projected DB. Then, *maxwp*(NX) = 0.5 + 0.8 = 1.3<*minsupport*, *maxwp*(YX) = 0.5 + 1.44 = 1.94 < minsupport. Thus, Y and N are removed from the {X}-projected DB. Then, item M (with *maxwp*(M) = 3.8) is handled. Dfimud extracts from two paths-names: (1) ⟨(N, 1.0),(Y, 1.0)⟩ occurs with (M, 1.0), and (2) ⟨(N, 2.8)⟩ occurs with (M, 2.8) and forms the {M}-projected DB. Then, *maxwp*(NM) = 1.0 + 2.8 = 3.8>minsupport. Therefore, {NM} is a high weighted probability itemset. However, {YM} is not a high weighted probability itemset because *maxwp*{YM} = 1.0 < minsupport.

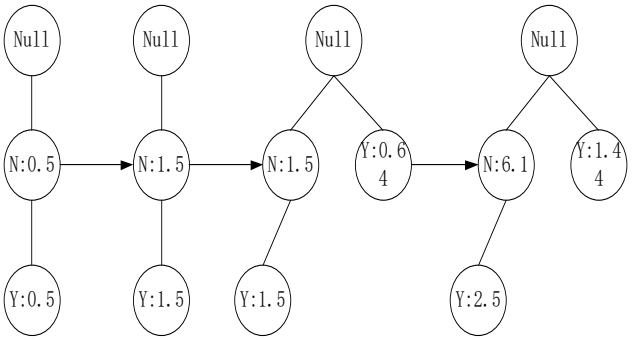


Figure 3. Process of constructing Dfimud-tree

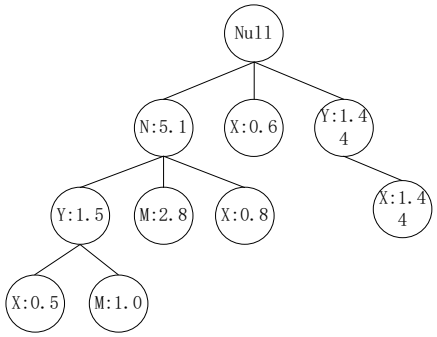


Figure 4. Dfimud-tree

The Dfimud only deals with items N and Y for group0 (and finds all high weighted probability itemsets) in a similar fashion. We find all high weighted probability itemsets as shown in Table 6.

Table 6. High weighted probability itemsets

| Group0 | Group1 |
|---|---|
| {N}:6.1 | {M}:3.8 |
| {Y}:3.94 | {X}:3.34 |
| | {NM}:3.8 |

According to Equation (1), $w$(M) = 0.9, $w$(MN) = ($w$(M) + $w$(N))/2 = (0.9 + 1.0)/2 = 0.95. According to Equation (2), $p$(M, $t_5$) = 0.8, $p$(M, $t_6$) = 0.8, $p$(M, $t_8$) = 0.6, $p$(M, $t_{10}$) = 0.7, $p$(MN, $t_5$) = 0.8 × 1.0 = 0.8, $p$(MN, $t_6$) = 0.8 × 1.0 = 0.8, $p$(MN, $t_8$) = 0.6 × 0.8 = 0.48, and $p$(MN, $t_{10}$) = 0.7 × 0.4 = 0.28. According to Equation (3), *expsup*(M) = $p$(M, $t_5$) + $p$(M, $t_6$) + $p$(M, $t_8$) + $p$(M, $t_{10}$) = 0.8 + 0.8 + 0.6 + 0.7 = 2.9 and *expsup*(MN) = $p$(MN, $t_5$) + $p$(MN, $t_6$) + $p$(MN, $t_8$) + $p$(MN, $t_{10}$) = 0.8 × 1.0+ 0.8 × 1.0+ 0.48 × 0.8+ 0.28 × 0.4= 2.36. According to Equation (4), *ewsup*(M) = $w$(M) × *expsup*(M) = 0.9 × 2.9 = 2.61 and *ewsup*(MN) = $w$(MN) × *expsup*(MN) = 0.95 × 2.36 = 2.242. Moreover, the *ewsup*(x) of each high weighted probability itemset is {N:4.45, Y:2.24, M:2.61, X:0.78, MN:2.242}. Then, all frequent patterns whose *ewsup*($x$) is no less than the minsupport are filtered out, that is, *ewsup*(x) ≥ $\varepsilon \times |T|$ = 3. Thus, the frequent pattern is {N, 4.45}.

## 4. Experimental Results and Analysis

### *4.1. Programming Environment and Datasets*

We depict the various experiments and the performance evaluation of Dfimud in this section. We compare Dfimud with the existing Uapriori algorithm (for mining expected support frequent itemsets in an uncertain dataset) and the existing PWA algorithm (for mining the weighted frequent itemsets in a certain dataset). We set up a Spark cluster of 12 nodes to implement the Dfimud algorithm. The Spark cluster consists of one master node and 11 slave nodes, each of which were allocated 4GB of RAM and two CPU cores. The Scala language is used to program. We store the input datasets and output results in HDFS. We use three benchmark datasets from the UCI machine learning repository to perform the experiments. The characteristics of the datasets are depicted in Table 7, where $|T|$ is the total number of transactions and $|L|$ is the average transaction length.

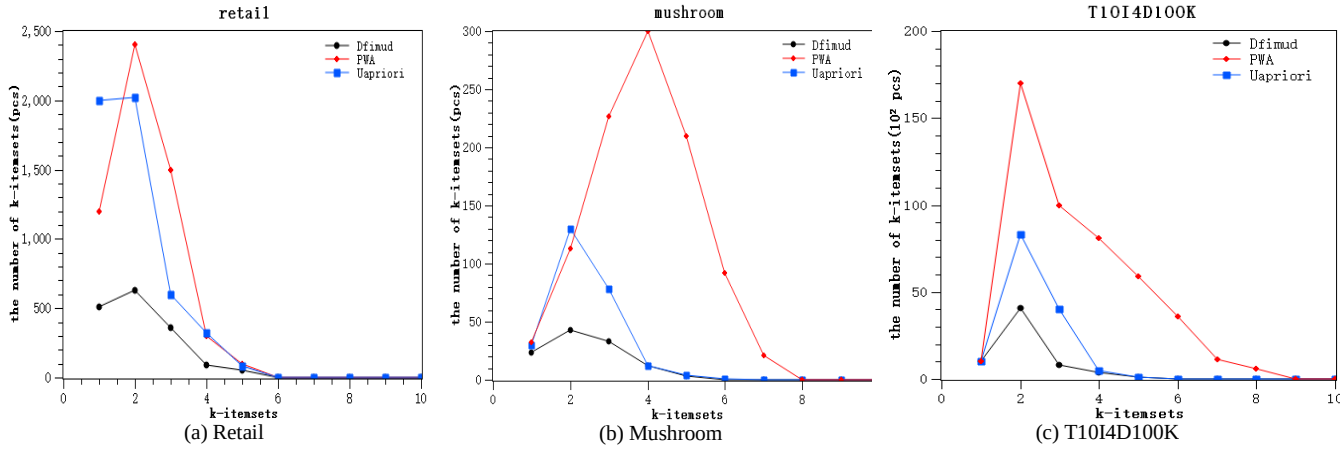
Table 7. Datasets characteristics

| Datasets | $\|T\|$ | $\|L\|$ |
|---|---|---|
| retail | 88162 | 10.3 |
| mushroom | 8124 | 23 |
| T10I4D100K | 100000 | 10.1 |

### *4.2. Experimental Results*

**Experiment 1** Patterns Analysis

The number of *k*-itemsets discovered by compared algorithms is evaluated. The results are shown in Figure 5. Here, we predefine 0.08% as the threshold for retail, 20% as the threshold for mushroom, and 0.03% as the threshold for T10I4D100K.




(a) Retail (b) Mushroom (c) T10I4D100K

Figure 5. The number of k-itemsets of different algorithms

From Figure 5, it can be observed that the frequent patterns number of Dfimud is generally less than the frequent pattern number of Uapriori and PWA. This is because the proposed algorithm finds frequent patterns by considering both the probability and weight measures. The Uapriori algorithm considers the expected support constraint, but it does not consider the weight support constraint. The PWA algorithm only considers the expected weight support constraint to discover frequent patterns. Thus, it is reasonable to discover fewer frequent patterns. For the frequent patterns number of PWA shown in Figure 5(a), 1-itemsets are fewer, but more *k*-itemsets ($k \geq 2$) are discovered. According to Figures 5(b) and 5(c), the number of 1-itemsets that are frequent patterns of Uapriori is close to the frequent pattern number of PWA, but the *k*-itemsets ($k \geq 2$) are much fewer than the frequent patterns of PWA. This is reasonable because the downward closure property of the final derived patterns does not hold in PWA algorithms. In the PWA algorithm, the weight support calculation method involves adding the weights of each item in the set and then dividing by the number of its item. This method may result in the weight of an item being less than the minimum support, but the weight of its superset is greater than the minimum support. Therefore, the downward closure property is not suitable in the PWA algorithm. In the proposed Dfimud algorithm, when considering both the weight and probability properties, fewer frequent patterns are produced. This is because as larger itemsets are explored, the probability that these *k*-itemsets occur becomes smaller. Thus, the frequent patterns of Dfimud are not only much fewer than the weighted or expected support patterns, but also more precise than the other patterns. Fewer and more meaningful patterns are thus discovered by the proposed algorithm. As more constraints are applied to patterns mining, more meaningful results and fewer patterns are found, indicating that the proposed novel method is reasonable and acceptable.

**Experiment 2** Performance of Runtime

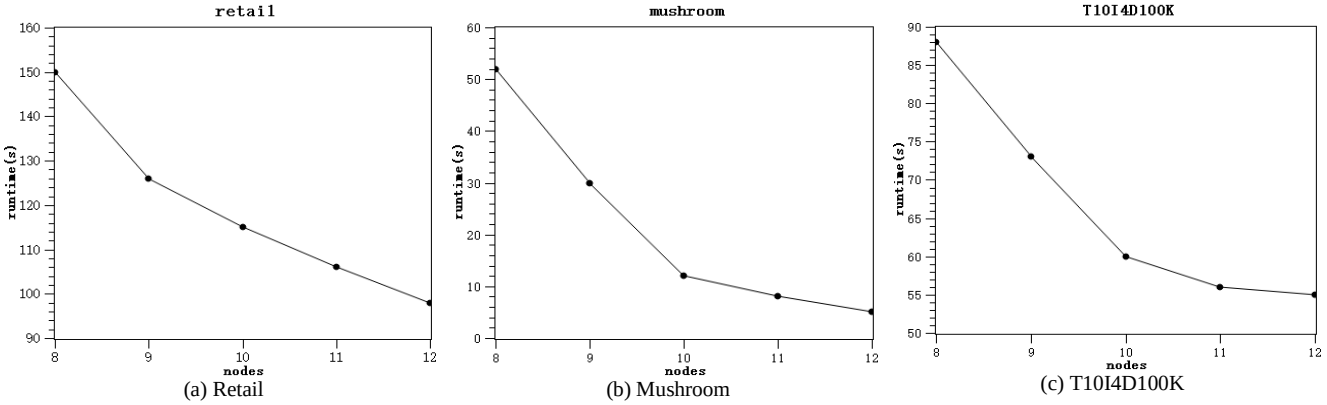
In this section, we analyse the execution time for Dfimud by changing the number of nodes in the cluster, as shown in Figure 6. We range the node count from 8 to 12 in the cluster and run the Dfimud for all three datasets. We set the threshold values as 0.08% for retail, 0.03% for T10I4D100K, and 20% for mushroom.

We observe that as we increase number of nodes, the execution time is reduced in an almost linear way. Hence, Dfimud obtains a near-linear performance for the scalability of nodes.

**Experiment 3** Algorithm Speedup

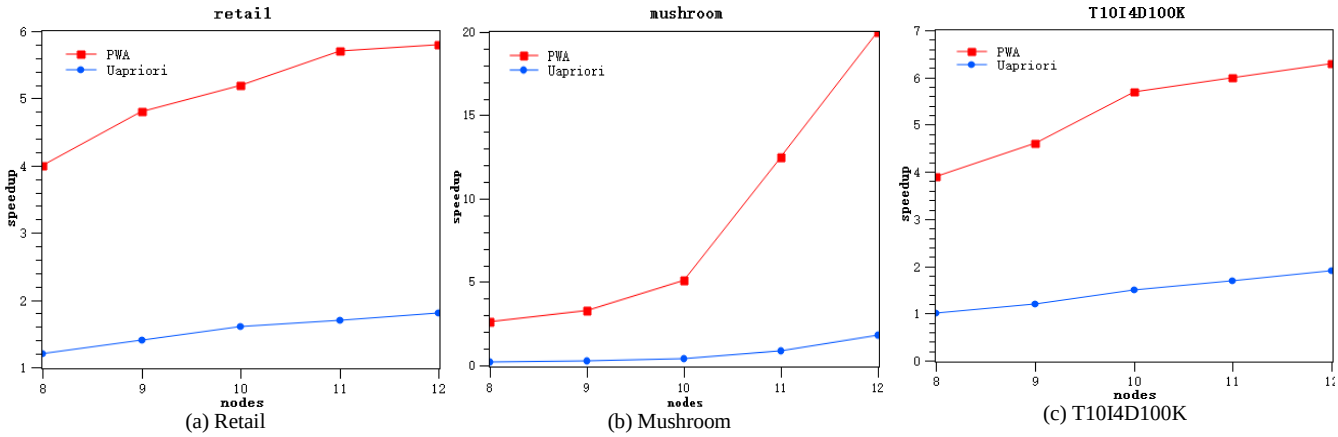
In order to examine the efficiency, the proposed Dfimud is compared with the traditional single on-machine algorithm and tested on UCI datasets, the speedup of which is the algorithm in a stand-alone environment and distributed environment running time ratio. We test the speedup performance of Dfimud by changing the number of nodes in the cluster, as shown in Figure 7. We range the node count from 8 to 12 in the cluster and run the Dfimud for all three datasets. The minimum support thresholds are 0.08% for retail and 0.03% for T10I4D100K. The results are depicted in Figure 7.

We observe that the speedup increases as the number of nodes increases. As shown in Figure 7(a), the speedup follows a steady trend when the number of nodes is set from 11 to 12; this is because the database size of retail is not enough to take advantage of the distributed algorithm. The speedup in Figure 7(c) always increases. The size of T1014100K is larger than retail, and advanced features of Spark can deal with large data efficiently and optimize the performance. We observe that the speedup is less than one when the number of nodes increases from 8 to 11 in Figure 7(b), so the runtime of Dfimud is not reduced under the distributed environment. This is because the mushroom database is a dense database. Dfimud needs to consider both the probability and weight of the itemsets, and fewer patterns are pruned when the minimum support threshold is small. This results in increased computational time. Overall, the proposed Dfimud has excellent performance and better scalability compared with the traditional method.




(a) Retail (b) Mushroom (c) T10I4D100K

Figure 6. Runtime for different numbers of nodes




(a) Retail (b) Mushroom (c) T10I4D100K

Figure 7. Speedup comparison of different scale datasets

## 5. Conclusions

In this paper, a Dfimud algorithm with FP-Growth feature is proposed for frequent pattern mining of massive uncertain data. Dfimud first devised the Dfimud-tree structure to store the conditional pattern base and other related information of itemsets in advance, reducing the number of the datasets scan times to one. Next, it pruned in advance, and a large number of candidate sets were avoided. Finally, Dfimud made good use of the efficient computing power of the Spark distributed platform and increased efficiency. Experiments on publicly available UCI datasets demonstrated that Dfimud achieves more optimal results than other related approaches across various metrics. In addition, the empirical study also showed that Dfimud has good scalability.

**Acknowledgements**

The paper is supported by the National Natural Science Foundation of China (No. 51467007, 61562054, and 61462050).

**References**

1. Y. Song, H. Z. Wang, J. Z. Li, and H. Gao, "MapReduce for Big Data Analysis: Benefits, Limitations and Extensions," in *Proceedings of International Conference of Pioneering Computer Scientists, Engineers and Educators*, pp. 453-457, 2016
2. Ö. M. Soysal, E. Gupta, and H. Donepudi, "A Sparse Memory Allocation Data Structure for Sequential and Parallel Association Rule Mining," *The Journal of Supercomputing*, Vol. 72, No. 2, pp. 347-370, 2016
3. C. W. Lin, W. Gan, P. Fournier-Viger, and T. P. Hong, "Efficient Mining of Weighted Frequent Itemsets in Uncertain Databases," in *Proceedings of 12th International Conference*, pp. 236-250, Springer International Publishing, 2016
4. M. R. Karim, M. Cochez, O. D. Beyan, C. F. Ahmed, and S. Decker, "Mining Maximal Frequent Patterns in Transactional Databases and Dynamic Data Streams: A Spark-based Approach," *Information Sciences*, Vol. 432, pp. 278-300, 2018
5. C. K. Chui, B. Kao, and H. Edword, "Mining Frequent Itemsets from Uncertain Data," in *Proceedings of 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Vol. 4426, pp. 47-58, Springer-Verlag, Nanjing, China, 2007
6. C. H. Chee, J. Jaafer, A. Izzatdin, M. H. Hasan, and W. Yeoh, "Algorithms for Frequent Itemset Mining: A Literature Review," *Artificial Intelligence Review*, Vol. 36, No. 3, pp. 1-9, 2018
7. L. Wang, D. W. Cheung, R. Cheung, S. D. Lee, and X. Yang, "Efficient Mining of Frequent Item Sets on Large Uncertain Datasets," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 24, No. 12, pp. 2170-2183, 2012
8. X. Sun, L. Lim, and S. Wang, "An Approximation Algorithm of Mining Frequent Itemsets from Uncertain Dataset," *International Journal of Advancements in Computing Technology*, Vol. 4, No. 3, pp. 42-49, 2012
9. C. K. Leung, C. L. Carmicheal, and B. Hao, "Efficient Mining of Frequent Patterns from Uncertain Data," in *Proceedings of 17th IEEE International Conference on Data Mining Workshops*, pp. 489- 494, IEEE Computer Society, Washington, DC, USA, 2007
10. W. L. Chun and P. H. Tzung, "A New Mining Approach for Uncertain Datasets using CUFP-Trees," *Expert Systems with Applications*, Vol. 39, No. 4, pp. 4084-4093, 2012
11. T. Bernecker, H. P. Kriegel, M. Renz, F. Verhein, and A. Zufle, "Probabilistic Frequent Itemset Mining in Uncertain Datasets," in *Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 119-128, ACM, Paris, France, 2009
12. L. Sun, R. Cheng, D. W. Cheung, and J. Cheng, "Mining Uncertain Data with Probabilistic Guarantees," in *Proceedings of 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 273-282, New York, NY, USA: Kdd, 2010
13. U. Yun and J. J. Leggett, "WFIM: Weighted Frequent Itemset Mining with a Weight Range and a Minimum Weight," in *Proceedings of Siam International Conference on Data Mining*, pp. 636-640, 2005
14. U. Yun and J. Leggett, "WSpan: Weighted Sequential Pattern Mining in Large Sequential Datasets," in *Proceedings of 3th International IEEE Conference on Intelligent Systems*, pp. 512-517, IEEE, London, UK, 2006
15. G. C. Lan, T. P. Hong, Y. L. Hong, S. L. Wang, and C. W. Tsai, "Enhancing the Efficiency in Mining Weighted Frequent Itemsets," in *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1104-1108, 2013
16. C. W. Lin, W. Gan, P. Fournier-Viger, and T. P. Hong, "RWFIM: Recent Weighted-Frequent Itemsets Mining," *Engineering Applications of Artificial Intelligence*, Vol. 45, pp. 18-32, 2015
17. C. W. Lin, W. Gan, P. Fournier-Viger, H. C. Chao, and T. P. Hong, "Efficiently Mining Frequent Itemsets with Weight and Recency Constraints," *Applied Intelligence*, Vol. 47, No. 3, pp. 769-792, 2017
18. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Weighted Frequent Itemset Mining over Uncertain Datasets," *Applied Intelligence*, Vol. 44, No. 1, pp. 232-250, 2016
19. H. Qiu, R. Gu, C. Yuan, and Y. Huang, "YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark," *IEEE International Parallel and Distributed Processing Symposium Workshops*, IEEE Computer Society, Washington, DC, USA, pp. 1664-1671, 2014
20. S. Rathee, A. Kashyap, and M. Kaul, "R-Apriori: An Efficient Apriori based Algorithm on Spark," *The 8th Workshop on Ph.D. Workshop in Information and Knowledge Management*, pp. 27-34, ACM, Melbourne, Australia, 2015
21. K. K. Sethi and D. Ramesh, "HFIM: A Spark-based Hybrid Frequent Itemset Mining Algorithm for Big Data Processing," *Journal of Supercomputing*, Vol. 73, pp. 1-17, 2017

**Jiaman Ding** is an associate professor in the Faculty of Information Engineering and Automation at Kunming University of Science and Technology. His current research interests include data mining, cloud computing, and machine learning.

**Haibin Li** is a MPhil student in the Faculty of Information Engineering and Automation at Kunming University of Science and Technology. His current research interests include machine learning and data mining.

**Yang Yang** is a MPhil student in the Faculty of Information Engineering and Automation at Kunming University of Science and Technology. Her current research interests include machine learning and data mining.

**Lianyin Jia** is an associate professor in the Faculty of Information Engineering and Automation at Kunming University of Science and Technology. He received his Ph.D. in computer science from South China University of Technology in 2013. His current research interests include databases, data mining, information retrieval, and parallel computing.

**Jinguo You** is an associate professor in the Faculty of Information Engineering and Automation at Kunming University of Science and Technology. He received his Ph.D. in computer science from South China University of Technology in 2009. His current research interests include data warehouses and data mining.