ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 1891–1900

Procedia Computer Science

www.elsevier.com/locate/procedia

22nd International Conference on Knowledge-Based and
Intelligent Information & Engineering Systems

Item-centric mining of frequent patterns from big uncertain data

Peter Braun[a], Alfredo Cuzzocrea[b], Carson K. Leung[a,*], Adam G.M. Pazdor[a], Joglas Souza[a]

[a] University of Manitoba, Winnipeg, MB, R3T 2N2, Canada
[b] University of Trieste, 34127 Trieste (TS), Italy

Abstract

High volumes of wide varieties of valuable data of different veracity (e.g., imprecise and uncertain data) can be easily generated or collected at a high velocity for various knowledge-based and intelligent information & engineering systems in many real-life situations. Embedded in these big data is valuable knowledge and useful information, which can be discovered by data science solutions. As a popular data science task, frequent pattern mining aims to discover implicit, previously unknown and potentially useful information and valuable knowledge in terms of sets of frequently co-occurring items. Many of the existing frequent pattern mining algorithms use a *transaction-centric* mining approach to find frequent patterns from *precise* data. However, there are situations in which an item-centric mining approach is more appropriate, and there are also situations in which data are imprecise and uncertain. In this article, we present an *item-centric* algorithm for mining frequent patterns from big *uncertain* data. Evaluation results show the effectiveness of our algorithm in item-centric mining of frequent patterns from big uncertain data.

Keywords: Big data; data analytics; data mining; data science; frequent patterns; imprecise and uncertain data; knowledge discovery in databases; uncertainty; vertical mining

1. Introduction

As technology advances, high volumes of a wide variety of data of different veracity (e.g., uncertain data [25, 31] in social networks [4, 24]) can be easily generated or collected at a high velocity from rich sources of data in various real-life big data applications and services. Embedded in these big data are useful information and valuable knowledge. Hence, *data science* [33] is in demand. In general, a data science solution applies the following to big data for *data analytics* [5, 22]:

- data mining [10, 41],

[*] Corresponding author.
E-mail address: kleung@cs.umanitoba.ca

1877-0509 © 2018 The Authors. Published by Elsevier Ltd.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0/)
Selection and peer-review under responsibility of KES International.
10.1016/j.procs.2018.08.075

- database [18],
- machine learning [32],
- mathematical modelling [3],
- statistical techniques [37], and/or
- visualization [15].

Among popular data science tasks, *frequent pattern mining* [2, 36] aims to discover sets of frequently co-occurring items. The mined patterns help data scientists understand consumer behavior, know the web surfing patterns, and get an insight about popular events. In other words, the mined patterns help the data scientists to get better understanding of the behaviors of their users.

Early frequent pattern mining algorithms (e.g., Apriori [2]) mostly execute the mining process in serial on a single local computer. To improve algorithmic efficiency, later algorithms perform the computation in distributed and parallel environments by using computer clusters or grids [14]. As volumes of big data can go beyond the ability of commonly-used software to capture, manage, and process within a tolerable elapsed time, new forms of processing, mining and analyzing data—such as MapReduce-based frequent pattern mining algorithms [8] that usually run in clouds [12]—have been developed to enable enhanced decision making, insight, knowledge discovery, and process optimization. In general, cloud computing makes use of a network of remote servers hosted on the Internet to store, manage, process, and mine the data.

With the aforementioned traditional databases of precise data, users definitely know whether an item is present in (or is absent from) a transaction. In contrast, data in many real-life applications are riddled with uncertainty [28, 34, 39]. It is partially due to the following:

- inherent measurement inaccuracies,
- sampling and duration errors,
- network latencies [13], and/or
- intentional blurring of data to preserve anonymity.

As such, the presence or absence of items in a dataset is uncertain. Moreover, with the increasing number of uncertain objects for sensor devices and noisy data management technologies such as DUST [35] in recent years, uncertain data-mining [6, 40] is in demand. As a concrete example, a physician may highly suspect (but cannot guarantee) that a coughing patient suffers from a cold. The uncertainty of such suspicion can be expressed in terms of existential probability (e.g., a 60% likelihood of suffering from the cold). With this notion, each item in a transaction t_j in databases containing precise data can be viewed as an item with a 100% likelihood of being present in t_j. To find frequent itemsets from these uncertain data, several uncertain data-mining algorithms (e.g., U-Apriori [11] and UF-growth [31] algorithms) have been proposed. Among them, UF-growth captures the contents of the uncertain data in a tree structure, from which frequent itemsets can be mined recursively.

Most (serial, parallel, and/or MapReduce-based) algorithms find frequent patterns from precise data rather than imprecise or uncertain data. For those that handles imprecise or uncertain data, they mine the data "horizontally" (i.e., in a transaction-centric fashion). However, there are situations in which it is more appropriate to mine the data "vertically" (i.e., in an item-centric fashion). Our *key contribution* of this article includes our algorithm for mining frequent patterns "vertically" from big uncertain data in an item-centric fashion. Evaluation results show that effectiveness of our algorithm.

The remainder of this article is organized as follows. Next section reviews some related works. Section 3 describes our algorithm for mining frequent patterns "vertically" from big uncertain data in an item-centric fashion. Section 4 shows our evaluation. Finally, conclusions are drawn in Section 5.

2. Related works

As the first step of association rule mining [19, 29], ***frequent pattern mining*** aims to discover sets of frequently co-occurring items. Many frequent pattern mining algorithms [20, 21] were Apriori-based [2], which rely on a generate-and-test paradigm to discover frequent patterns from transaction datasets by first generating candidates and then checking their actual frequency (i.e., occurrences) against the datasets. To improve algorithmic efficiency, other serial frequent pattern mining algorithms (e.g., tree-based algorithms [16]) have been developed. In addition, there are also distributed and parallel algorithms [30, 38]. They find frequent patterns by using multiple processors that have either of the following:

- access to a shared memory with which information is exchanged among processors, or
- their own private memory (i.e., distributed memory) with which for information is exchanged via message passing among the processors.

These algorithms are mainly implemented using Open Multi-Processing (OpenMP) or Message Passing Interface (MPI), and run in computer clusters or grids. Cluster computing usually involves a group of distributed or parallel computers that are interconnected through high-speed networks such as local area networks; grid computing coordinates heterogeneous networked loosely coupled computers.

For precise data mining, while there are algorithms that view the transaction database TDB "horizontally" (e.g., Apriori, FP-growth), there are also algorithms (e.g., Eclat [42]) that view the TDB "vertically" [42] as a collection of items (i.e., TDB = $\{tidset(x) \mid \forall x \in$ domain items$\}$), where a tidset is a set of transaction IDs. Each item is associated with a set or vector that indicates the transactions containing such an item. For example, TDB $=\{tidset(a), tidset(b), \ldots\}$, and $tidset(a)=\{t_1, t_3, t_4, t_5\}$ representing that item a appears in four transactions t_1, t_3, t_4 and t_5. More specifically, Eclat first uses a set of transaction IDs (i.e., tidset) for every domain items to represent all the transactions containing x, and then finds frequent patterns in a bottom-up fashion. A pattern X is frequent if its actual support (i.e., its number of occurrence) in the TDB meets the user-specified threshold *minsup*. Although Eclat mines frequent patterns from the TDB, it does not mine the uncertain data.

To handle big data, MapReduce-based frequent pattern mining algorithms [8] read input data, divide them into several partitions (sub-problems), and assign them to different processors. Each processor executes the "map" function on each partition by taking a pair of $\langle key_1, value_1\rangle$ and returning a list of $\langle key_2, value_2\rangle$-pairs as an intermediate result, which is then shuffled and sorted:

$$\text{map: } \langle key_1, value_1\rangle \mapsto \text{list of } \langle key_2, value_2\rangle$$

where

- key_1 and key_2 are keys on some domains; and
- $value_1$ and $value_2$ are the coressponding values associated with key_1 and key_2, respectively.

Each processor then executes the "reduce" function on a single key_2 from this intermediate result by combining, aggregating, summarizing, filtering, and/or transforming the list of values associated with a given key key_2 (for all keys). As a result, a single (aggregated or summarized) value $value_3$ is returned:

$$\text{reduce: } \langle key_2, \text{list of } value_2\rangle \mapsto \text{list of } \langle key_3, value_3\rangle$$

where

- key_2 and key_3 are keys on some domains; and
- $value_2$ and $value_3$ are the coressponding values associated with key_2 and key_3, respectively.

Many of these algorithms are run in (public, private, or hybrid) clouds. Cloud computing [7] involves a group of interconnected and virtualized computers to provide on-demand services such as the following:

- function-as-a-service (FaaS),
- infrastructure-as-a-service (IaaS),
- network-as-a-service (NaaS),
- platform-as-a-service (PaaS),
- software-as-a-service (SaaS), and

- mining-as-a-service (MaaS) [17].

The research problem of frequent itemset mining was first introduced [2] in 1993. The corresponding algorithm—namely, Apriori—mined all frequent itemsets from a transaction database (TDB) consisting of precise data, in which the contents of each transaction are precisely known. Specifically:

- If a transaction t_i contains an item x (i.e., $x \in t_i$), then x is precisely known to be present in t_i.
- On the other hand, if a transaction t_i does not contain an item y (i.e., $y \notin t_i$), then y is precisely known to be absent from t_i.

However, this is not the case for probabilistic databases consisting of uncertain data [9]. A key difference between precise and uncertain data is that each transaction of the latter contains items and their existential probabilities. The existential probability $P(x, t_i)$ of an item x in a transaction t_i indicates the likelihood of x being present in t_i. For a real-life example, each transaction t_i represents a patient's visit to a physician's office. Each item x within t_i represents a potential disease, and is associated with $P(x, t_i)$ expressing the likelihood of a patient having that disease x in t_i (say, in t_1, the patient has a 60% likelihood of having asthma, and a 90% likelihood of catching a cold regardless of having asthma or not). With this notion, each item in a transaction t_i in datasets of precise data can be viewed as an item with a 100% likelihood of being present in t_i. Given an item x and a transaction t_i, there are two possible worlds when using the possible world interpretation of uncertain data [1, 26]:

1. The possible world W_1 where $x \in t_i$, and
2. the possible world W_2 where $x \notin t_i$.

Although it is uncertain which of these two worlds would be the true world, the probability of any one of them being the true world can be expressed as follows:

- the probability of W_1 being the true world is $P(x, t_i)$, and
- the probability of W_2 being the true world is $1 - P(x, t_i)$.

Let (i) Item be a set of m domain items and (ii) $X = \{x_1, x_2, \ldots, x_k\}$ be a k-itemset (i.e., a pattern consisting of k items), where $X \subseteq$ Item and $1 \leq k \leq m$. Then, a transactional database is the set of n transactions, where each transaction $t_j \subseteq$ Item (for $1 \leq j \leq n$). The projected database of X is the set of all transactions containing X. Each item x_i in a transaction $t_j = \{x_1, x_2, \ldots, x_h\}$ in an uncertain database is associated with an *existential probability* $P(x_i, t_j)$, which represents the likelihood of the presence of x_i in t_j [26], with value:

$$0 < P(x_i, t_j) \leq 1 \tag{1}$$

The existential probability $P(X, t_j)$ of a pattern X in t_j is then the product of the corresponding existential probabilities of every item x_i within X when these items are independent [25, 27]:

$$P(X, t_j) = \prod_{x_i \in X} P(x_i, t_j) \tag{2}$$

Finally, the *expected support* *expSup*(X) of X is the sum of $P(X, t_j)$ over all n transactions in the database:

$$expSup(X) = \sum_{j=1}^{n} P(X, t_j) = \sum_{j=1}^{n} \left(\sum_{x_i \in X} P(x_i, t_j) \right) \tag{3}$$

With this notion of expected support, existing tree-based algorithms—such as UF-growth [31]—mine frequent patterns from uncertain data by first scanning the uncertain database once to compute the expected support of all domain items (i.e., singleton patterns). Infrequent items are pruned as their extensions/supersets are guaranteed to be infrequent. The algorithms then scan the database a second time to insert all transactions (with only frequent items) into a tree (e.g., UF-tree [31]). Each node in the tree captures the following:

- an item x,
- its existential probability $P(x, t_j)$, and
- its occurrence count.

At each step during the mining process, the frequent patterns are expanded recursively.

A pattern X is *frequent* in an uncertain database if $expSup(X) \geq minsup$. Given a database and *minsup*, the research problem of frequent pattern mining from uncertain data is to discover from the database a complete set of frequent patterns having expected support ≥ *minsup*.

3. Our Spark-based vertical mining algorithm

In this section, let us first describe how we handle uncertain data, then describe how we mine these uncertain data in the Spark environment.

3.1. Handle uncertain data

When mining precise data, Eclat uses a tidset for each domain item x to represent all the transactions containing x, and finds frequent patterns from the transaction database (TDB) in a bottom-up candidate generation fashion. Here, when mining uncertain data, it is important to capture the existential probability $P(x, t_i)$ associated with each item x in each transaction t_i of the uncertain database (UDB).

Our first issue is *how to represent the UDB that contains these existential probability values?* To deal with this issue, it may be tempting to use a tidset for each distinct ⟨domain item, existential probability value⟩-pair. However, a potential problem of such an attempt is the huge number of pairs. Theoretically, it could be an uncountably infinite number because the existential probability could take on a real number in the range of (0, 1]. It is fortunate that, practically, the number of such pairs is countable and bounded above by the number of ⟨domain item, existential probability value⟩-pairs present in the UDB. Hence, we aim to find patterns having their expected support meeting the user-specified threshold. In other words, we want to find every frequent pattern X (and its expected support) rather than the occurrence count of every ⟨domain item, existential probability value⟩-pair. As such, we do not need to keep a tidset for each distinct ⟨domain item, existential probability value⟩-pair. Instead, it is more logical to use a tidset for each domain item x—than ⟨x, existential probability⟩-pairs—in our proposed algorithm. Specifically, *it augments to each tidset (for x) the corresponding existential probability values associated with x in the transactions.*

The next issue is *how to compute the expected support of a singleton pattern (e.g., item a)?* When mining precise data, one can compute the actual support of an item x by knowing the cardinality of $tidset(x)$ because the actual support of x counts the number of transactions in which x occurs. However, the expected support of x cannot be obtained by the cardinality of $atidset(x)$. For example, $|atidset(\{c\})| = 4$, but $expSup(\{c\})$=2.9. The reason is that, for precise data, each occurrence of x leads to an increment of 1 to its support count; for uncertain data, each occurrence of x does not lead to an increment of 1 to its expected support value. An item x may occur twice, but if its existential probability is low, then its expected support may be lower than another item y that only occurs once but has high existential probability (e.g., x:0.1 appearing twice leads to $expSup(x)=0.2 < 0.5 = expSup(y)$ for y:0.5 appearing only once). For uncertain data, existential probability of x (instead of its occurrence) contributes to its expected support value. Hence, *we compute the expected support of an item x by summing its existential probability*:

$$expSup(x) = \sum_{t_i} \{P(x, t_i) | \langle t_i{:}P(x, t_i)\rangle \in atidset(x)\} \tag{4}$$

Once we computed the expected support of singleton patterns and found the frequent ones, the next issue is *how to generate candidate patterns of higher cardinalities (e.g., k-itemsets consisting of k items each)?* When mining precise data, one can generate candidate k-itemsets (say, $\alpha \cup \{x, y\}$, or αxy for short) by intersecting the tidsets of two frequent $(k-1)$-itemsets, i.e., $tidset(\alpha x) \cap tidset(\alpha y)$. However, when mining uncertain data, applying a similar approach that $expSup(\alpha xy\}) = |atidset(\alpha x) \cap atidset(\alpha y)|$ may lead to incorrect results because $atidset(Z)$ of any itemset Z is a set of $\langle t_i{:}expSup(Z, t_i)\rangle$-pairs (instead of a set of just t_i). Intersecting the two augmented tidsets is more restrictive than what we want. Note that intersecting the two augmented tidsets returns those transactions containing $\alpha x = \alpha y$ and having the same existential probability value. However, what we want are transactions containing both αx and αy regardless of their existential probability values. Hence, it is better to generate candidates by intersecting only the tidset components of the augmented tidsets. The above approach partially solves the problem. Specifically, it gives the intended set of transaction IDs for the intersection. However, it may still lead to inaccurate expected support value for the resulting candidate. To completely solve this problem, *we generate a candidate k-itemset by intersecting tidsets of a frequent* $(k-1)$*-itemset and a frequent 1-itemset*, as shown below:

$$\begin{aligned} atidset(\alpha xy) = \{&\langle t_i : expSup(\alpha x, t_i) \times P(y, t_i)\rangle | \\ &\langle t_i : expSup(\alpha x, t_i)\rangle \in atidset(\alpha x) \wedge \langle t_i : P(y, t_i)\rangle \in atidset(y)\} \end{aligned} \tag{5}$$

From $atidset(\alpha xy)$, we can then compute the expected support of αxy as follows:

$$\begin{aligned} expSup(X) &= \sum_{t_i} \{expSup(X, t_i) | \langle t_i : expSup(X, t_i) \in atidset(X)\} \\ &= \sum_{t_i} \left\{ \prod_{x \in X} P(x, t_i) | \langle t_i : expSup(X, t_i) \in atidset(X) \right\} \end{aligned} \tag{6}$$

To recap, in this section, we pointed out three issues related to handling uncertain data, and we provided the following solutions to each of these issues:

1. How to represent the UDB that contains these existential probability values?
   We augment to each $tidset(x)$ the corresponding existential probability value $P(x, t_i)$ associated with x in the transaction t_i.
2. How to compute the expected support of a singleton pattern x?
   We compute the expected support $expSup(x)$ of an item x by summing its existential probability: $expSup(x) = \sum_{t_i} \{P(x, t_i) | \langle t_i : P(x, t_i)\rangle \in atidset(x)\}$.
3. How to generate candidate patterns of higher cardinalities (e.g., k-itemsets consisting of k items each)?
   We generate a candidate k-itemset $\alpha \cup \{x, y\}$ by intersecting tidsets of a frequent $(k-1)$-itemset $\alpha \cup \{x\}$ and a frequent 1-itemset $\{y\}$. Then, from $atidset(\alpha \cup \{x, y\})$, we can then compute the expected support $expSup(\alpha \cup \{x, y\})$ of $(\alpha \cup \{x, y\})$ by $expSup(\alpha \cup \{x, y\}) = \sum_{t_i} \left\{ \prod_{z \in \alpha \cup \{x,y\}} P(z, t_i) | \langle t_i : expSup(\alpha \cup \{x, y\}, t_i) \in atidset(\alpha \cup \{x, y\}) \right\}$.

In other words, we perform the following steps when handling uncertain data:

1. We push uncertain mining inside the frequent pattern mining process by first building an augmented tidset for each domain item x to capture

   - every transaction t_i that contains x and
   - its existential probability $P(x, t_i)$ in that transaction.

2. Then, the expected support of a pattern X is computed by summing the expected support of X over all transactions in $atidset(X)$.
3. If $expSup(X)$ meets the user-specified threshold *minsup*, then X is frequent; otherwise, X is pruned as an infrequent pattern.
4. Candidate k-itemsets are generated by intersecting the tidsets of frequent $(k-1)$-itemsets with those of frequent 1-itemsets.
5. Repeat Steps 2–4 until no more candidate k-itemsets can be generated (i.e., no more frequent k-itemsets can be discovered).

3.2. Handle big uncertain data

Next, we describe how to extend the above (for handling uncertain data) to handle big data by using the Spark-based approach. Specifically, we scan the original database once and convert it into vertical representation. The algorithm only keeps track of items exist in transactions. In order to form new itemset, we intersect two vectors having those entries that have the same transaction IDs. When forming the vector $\vec{ab}$, we intersect vector $\vec{a}$ with vector $\vec{b}$. We eliminate (3, 0.3) in vector $\vec{a}$ as it can not find any transaction ID match in vector $\vec{b}$. As the result, we have ⟨*ab*, (1, 0.1×0.2), (2, 0.2×0.3)⟩. We observe that the following two key matching behaviors:

1. The item key need to be matched.
2. The transaction IDs have to be matched.

Hence, in our proposed MR-UV-Eclat algorithm, the key of the mapped record becomes "item|transactionID". Consequently, it reduces phase to map not only the item keys but also transactionIDs. Hence, this algorithm first convert original data to be the format in ⟨item|transactionID, value⟩. In the reduce phase, it collects data with the same key of "item|transactionID" and performs the multiplication.

For the way forming new candidates, we find the last entry of *b* and then base on the position of *b* presented in $\{a, b, c, d, e\}$, we find the subset of $\{c, d, e\}$. The candidate 3-itemsets generated from *ab* are formed by doing concatenation of *ab* with one of $\{c, d, e\}$. As the result, we get candidates of $\{abc, abd, abe\}$. We observe that the key matching process of the map-reduce behavior in MR-UV-Eclat is in transaction level.

Items are stored in a trie structure so as to speed up the search. Only frequent nodes continue to expend for next level's examinations. One of the main advantage of using a trie structure is that it can reduce candidate generation range.

As we try to match item key and transaction ID when doing reduce operation, the map key needs to contain item key and transaction ID. In addition, we need to store the support value on the corresponding the item and the specific transaction, this value is stored in the mapped key. Hence, the format of the key is like "itemKey|transactionID|supportValue".

What should store in the value of the mapped record? Trie. Each mapped record represent a node in the trie structure and each trie structure only represents a transaction. Thus, the value of each mapped record contains potential suffix items. Here, each suffix item not only contains the singleton but also its support value in the corresponding transaction. There is a "remap" process in each mining process. The "remap" process is as follow:

1. Form new key by concatenating item key with each of suffix item,
2. multiply the support value with each of uncertain value of each suffix item.

The potential suffix singletons with their support values are store in the mapped value in the mapped record. For example, the value for node *a* is (*b*:0.8, *c*:0.7, *d*:0.6). Notice that the node *d* has empty set in the mapped value because no item is after item *d*. In order to generate the next level of candidates, our algorithm concatenates the item key with each of suffix items. The support value for the new candidate in the current transaction is calculated by multiplying the support value with the uncertain value of the suffix item. For example, node *a* generates the following 3 nodes for next level because there are 3 suffix items in the mapped value:

1. $\{a, b\}$, with its support value $0.8 \times 0.8 = 0.64$;
2. $\{a, c\}$, with its support value $0.8 \times 0.7 = 0.56$; and
3. $\{a, d\}$, with its support value $0.8 \times 0.6 = 0.48$.

The suffix items for the each node are items after the visited item. For example, the suffix items for $\{a, b\}$ are (*c*:0.7, *d*:0.6), the suffix items for $\{a, c\}$ are (*d*:0.6), and no suffix item for $\{a, d\}$.

How to calculate the total frequency of the itemset? Through "remap" process, we expand each node to generate candidates and calculate their support values for specific transactions. In order to calculate the final support values for the candidates, we require the other "remap" process. We observe that each mapped key contains the itemset key, transaction ID and its support value. To get the final support value, we need to sum up these values with the same itemset key. This means that we need one more map-reduce process to conduct this calculation. First of all, we need to build up the other *resilient distributed dataset* (*RDD*) which is with the format of (*itemsetKey, supportValue*). We can get all these from just mapped key from RDD we just created before. Simply, we calculate the final support value in the reduce process.

3.2.1. Details

MR-UV-Eclat first reads each transaction of original database. When reading each transaction, it constructs tree projection for the first level. Each node becomes the mapped data. As this is for the level 1 data, the trie is only expanded to 1 level below "NULL" node. The value of each mapped record contains potential suffix items for the other level of expansion. After expansion is done, the next operation is frequency examination. This through the other map-reduce operation. First, the algorithm goes over each mapped key and put item key as new key, and support value as the new value. In the reduce phase, it sums up the frequency in all transactions. The final support values are compared with minimum support threshold. Only frequent items are kept in L and L_k.

After vertical data conversion, each node is free to expand. From the mapped key, we extract the current itemset, support value and transaction ID. The value of the mapped record is a list of nodes that each of them are potential suffix singletons with its support value in the current transaction. The tree expansion is to concatenate the current itemset with each item in the list. This new candidate is then verified by a pruning process with the input of L_{k-1}. If this candidate passes the examination of the pruning process, the new support value is then calculated by multiplying the current support value with the support value in a node in the list. The new candidate, the new support value and the current transaction ID together form the new mapped key. The new value of the mapped record is the list from the one after the current visited node until the end of the list. After this tree expansion, all these candidate nodes are generated. We still need to utilize map-reduce operation to conduct frequency examination. First, through the "map" process, we extract candidate itemset and its support value from the mapped records formed previously. Second, through the reduce process, the algorithm sums up all support value and form the final output format of (itemset, value). In between, the value of this mapped record is compared with minimum support threshold. Only ones that are large than minimum support threshold are kept. These data are then kept in L_k.

4. Evaluation

First, we evaluated our MR-UV-Eclat algorithm with the horizontal big data mining algorithm (say, a MapReduce version of U-Apriori [11]). We ran both algorithms in Spark, which mainly handles big data and enable in-memory computation. As most of data are stored on the hard disk in each worker in the master-worker environment. Most of in-memory computations happen in "map" and "reduce" processes. Each worker fetches data from its hard disk or from shuffles and then build RDD in memory. If the RDD cannot fit into memory, it will store them in temperate hard disk storage and bring them to main memory again when there are room available in main memory.

4.1. Analytical evaluation

In this memory usage evaluation, we examined the process on each algorithm that consumes the most memory usage. Specifically, we count how many nodes each algorithm create for its peak time (the time that the algorithm consumes the most memory). We find out that the reduce phase on each algorithm create nodes that at most the number of total transactions. We find out that the peak time for each algorithm is the mapping process on generating candidates. For this evaluation, we use the worst-case scenario approach to exanimate memory usage. We use m to denote the total number of items, k denote the current level of candidates. Then, the total number of nodes in the mapping phase of MR-U-Apriori is:

$$\sum_{j=1}^{k}\sum_{i=1}^{m} 2(m-j+1-(i-1)) = \sum_{j=1}^{k}\sum_{i=1}^{m} 2(m-j-i+2) \tag{7}$$

In contrast, as MR-UV-Eclat is based on the current level of itemsets to generate the next level candidates, the total number of nodes in the mapping phase of our MR-UV-Eclat is much smaller, as follows:

$$\sum_{i=1}^{m}(m-k+1-(i-1)) = \sum_{i=1}^{m}(m-k-i+2) \tag{8}$$

4.2. *Experimental evaluation*

In addition, experimental evaluation was also conducted. Fig. 1 shows that the runtime decreased with an increase of minsup for two datasets. MR-U-Apriori took much longer than our MR-UV-Eclat, especially for low *minsup*.

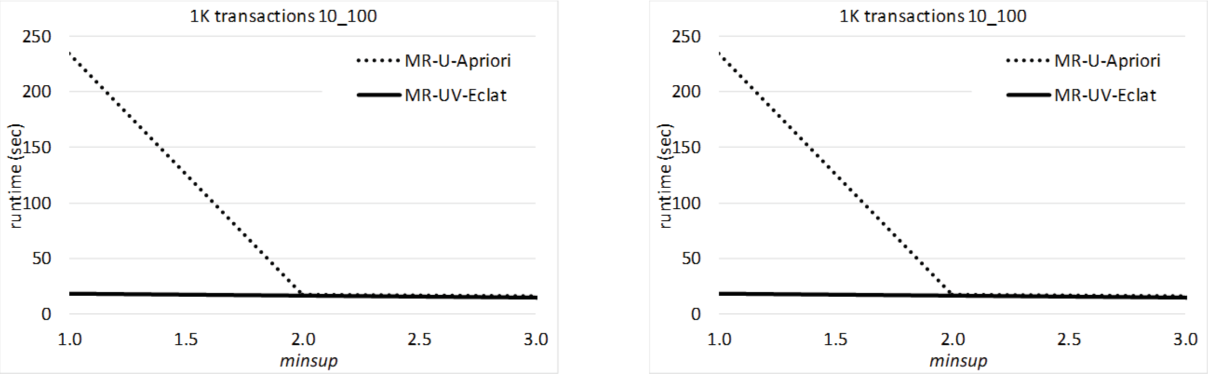

Fig. 1. Evaluation: runtime vs. *minsup*.

5. Conclusions

In this article, we presented an item-centric mining algorithm called MR-UV-Eclat for finding frequent patterns from big uncertain data in the Apache Spark environment. Both analyticsal and experimental evaluation results show its effectiveness and practicality.

As ongoing and future work, we are continue to conduct more extensive evaluation and explore more functional features to be provided to the users.

Acknowledgements

This project is partially supported by NSERC (Canada) and University of Manitoba.

References

[1] Aggarwal CC, Li Y, Wang J, Wang J. Frequent pattern mining with uncertain data. In: ACM KDD 2009, pp. 29-37.
[2] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: VLDB 1994, pp. 487-499.
[3] Akhmetzyanov AV, Salnikov AM. Mathematical models of unsteady flow distribution in gas transport systems. Procedia Computer Science 2017; 112: 1160-1167.
[4] Azaouzi M, Romdhane LB. An evidential influence-based label propagation algorithm for distributed community detection in social networks. Procedia Computer Science 2017; 112: 407-416.
[5] Balachandran BM, Prasad S. Challenges and benefits of deploying big data analytics in the cloud for business intelligence. Procedia Computer Science 2017; 112: 1112-1122.
[6] Bernecker T, Kriegel H, Renz M, Verhein F, Zuefle A. Probabilistic frequent itemset mining in uncertain databases. In: ACM KDD 2009, pp. 119-127.
[7] Bierzynski K, Escobar A, Eberl M. Cloud, fog and edge: cooperation for the future? In: FMEC 2017, pp. 62–67.
[8] Braun P, Cuzzocrea A, Jiang F, Leung CK, Pazdor AGM. MapReduce-based complex big data analytics over uncertain and imprecise social networks. In: DaWaK 2017, pp. 130-145.
[9] Cannataro M, Cuzzocrea A, Pugliese A. A probabilistic approach to model adaptive hypermedia systems. In: WebDyn 2001, pp. 50–60.
[10] Chen D, Sain SL, Guo K. Data mining for the online retail industry: a case study of RFM model-based customer segmentation using data mining. Journal of Database Marketing and Customer Strategy Management 2012; 19(3): 197-208.
[11] Chui C, Kao B, Hung E. Mining frequent itemsets from uncertain data. In: PAKDD 2007, pp. 47-58.

[12] Cuzzocrea A, Fortino G, Rana O. Managing data and processes in cloud-enabled large-scale sensor networks: state-of-the-art and future research directions. In: CCGRID 2013, pp. 583-588.

[13] Cuzzocrea A, Grasso GM, Jiang F, Leung CK. Mining uplink-downlink user association in wireless heterogeneous networks. In: IDEAL 2016, pp. 533-541.

[14] Cuzzocrea A, Leung CK, MacKinnon RK. Mining constrained frequent itemsets from distributed uncertain data. Future Generation Computer Systems 2014; 37: 117-126.

[15] Ghorbel F, Hamdi F, Ellouze N, Métais E, Gargouri F. Visualizing large-scale linked data with memo graph. Procedia Computer Science 2017; 112: 854-863.

[16] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1-12.

[17] Han Z, Leung CK. FIMaaS: scalable frequent itemset mining-as-a-service on cloud for non-expert miners. In: BigDAS 2015, pp. 84-91.

[18] Hassen F, Grissa-Touzi A. An efficient synchronous indexing technique for full-text retrieval in distributed databases. Procedia Computer Science 2017; 112: 811-821.

[19] Houari A, Ayadi W, Ben Yahia S. Mining negative correlation biclusters from gene expression data using generic association rules. Procedia Computer Science 2017; 112: 278-287.

[20] Lakshmanan LVS, Leung CK, Ng RT. Efficient dynamic mining of constrained frequent sets. ACM TODS 2003; 28(4): 337-389.

[21] Lakshmanan LVS, Leung CK, Ng RT. The segment support map: scalable mining of frequent itemsets. ACM SIGKDD Explorations 2000; 2(2): 21-27.

[22] Leung CK. Data and visual analytics for emerging databases. In: EDB 2017. LNEE, vol. 461, pp. 203-213 (2017)

[23] Leung CK. Frequent itemset mining with constraints. Encyclopedia of Database Systems, 2e; 2016. doi:10.1007/978-1-4899-7993-3_170-2

[24] Leung CK. Mathematical model for propagation of influence in a social network. Encyclopedia of Social Network Analysis and Mining, 2e; 2017. doi:10.1007/978-1-4614-7163-9_110201-1

[25] Leung CK. Mining frequent itemsets from probabilistic datasets. In: EDB 2013, pp. 137-148.

[26] Leung CK. Mining uncertain data. WIREs Data Mining and Knowledge Discovery 2011; 1(4): 316-329.

[27] Leung CK. Uncertain frequent pattern mining. In: Frequent Pattern Mining; 2014, pp. 417-453.

[28] Leung CK, Hao B. Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE 2009, pp. 1663-1670.

[29] Leung CK, Jiang F, Dela Cruz EM, Elango VS. Association rule mining in collaborative filtering. In: Collaborative Filtering Using Data Mining and Analysis; 2017, pp. 159-179.

[30] Leung CK, Jiang F, Pazdor AGM. Bitwise parallel association rule mining for web page recommendation. In: IEEE/WIC/ACM WI 2017, pp. 662-669.

[31] Leung CK, Mateo MAF, Brajczuk DA. A tree-based approach for frequent pattern mining from uncertain data. In: PAKDD 2008, pp. 653-661.

[32] Mercaldo F, Nardone V, Santone A. Diabetes mellitus affected patients classification and diagnosis through machine learning techniques. Procedia Computer Science 2017; 112: 2519-2528.

[33] Nimmagadda SL, Reiners T, Rudra A. An upstream business data science in a big data perspective. Procedia Computer Science 2017; 112: 1881-1890.

[34] Rahman MM, Ahmed CF, Leung CK, Pazdor AGM. Frequent sequence mining with weight constraints in uncertain databases. In: ACM IMCOM 2018, art. 48.

[35] Sarangi SR, Murthy K. DUST: a generalized notion of similarity between uncertain time series. In: ACM KDD 2010, pp. 383-392.

[36] Suzuki N, Matsuno H. Radio wave environment analysis at different locations based on frequent pattern mining. Procedia Computer Science 2017; 112: 1396-1403.

[37] Tanaka-Yamawaki M. Statistical distribution of the arrowhead price fluctuation. Procedia Computer Science 2017; 112: 1439-1447.

[38] Tanbeer SK, Ahmed CF, Jeong B. Parallel and distributed frequent pattern mining in large databases. In: IEEE HPCC 2009, pp. 407-414.

[39] Tong W, Leung CK, Liu D, Yu J. Probabilistic frequent pattern mining by PUH-Mine. In: APWeb 2015, pp. 781-793.

[40] Tong Y, Chen L, Cheng Y, Yu PS. Mining frequent itemsets over uncertain databases. PVLDB 2012; 5(11): 1650-1661.

[41] Veresnikov GS, Skryabin AV. The development of data mining methods criteria to identify failures of aircraft control surface actuator. Procedia Computer Science 2017; 112: 1007-1014.

[42] Zaki MJ, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: KDD 1997, pp. 283-286.