REDUCING THE SEARCH SPACE FOR INTERESTING PATTERNS FROM UNCERTAIN DATA

Dr. A.Meiappane [1], P.Ganesh Guru Teja [2], Meganathan.I [3] , Nilavazhagan.K [4]

[1] Associate Professor, M.Tech., Ph.D, Dept of Information Technology, MVIT, Puducherry,India
[234] B.Tech, Dept of Information Technology, MVIT, Puducherry,India

---***---

Abstract - *Big data is a term for referring too large and complex data sets that are difficult to process and store with the existing technologies. Data in the world is growing so fast in such a way that it will never be the same. The noise in that data is also increasing in the same manner, this causes a big problem in the mining. When searching in uncertain data it consists of the existential probabilities. User are only interested in an tiny portion of data. There exists many techniques for working with big data algorithms to mine the uncertain data but they are not giving satisfiable results. So we propose an algorithm which will be using the map reduce model with the apriori algorithm to mine the uncertain big datasets.*

***Key Words*: Big Data; Data Mining; MapReduce; Apriori; Search Space; Hadoop;**

1.INTRODUCTION

Big data is a broad term for datasets so large or complex that traditional data processing applications are inadequate. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying and information privacy. The term often refers simply to the use of predictive analytics or certain other advanced methods to extract value from data, and seldom to a particular size of data set. Accuracy in big data may lead to more confident decision making, and better decisions can result in greater operational efficiency, cost reduction and reduced risk. Analysis of data sets can find new correlations to spot business trends, prevent diseases, combat crime and so on. Scientists, business executives, practitioners of medicine, advertising and governments alike regularly meet difficulties with large data sets in areas including Internet search, finance and business informatics. Datasets are growing rapidly in part because they are increasingly gathered by cheap and numerous information-sensing mobile devices, aerial (remote sensing), software logs, cameras, microphones, Radio-Frequency Identification readers, wireless sensor networks. Every day 2.5 exabytes (2.5×1018) of data is created. One question for large enterprises is determining who should own big data initiatives that affect the entire organizations technology advances, high volumes of valuable data such as streams of banking, financial, marketing, telecommunication, biological, medical, life science, and social data are generated in various real-life applications in modern organizations and society. This leads us into the new era of Big data [18], which refer to interesting high-velocity, high value, and/or high-variety data with volumes beyond the ability of commonly-used software to capture, manage, and process within a tolerable elapsed time. Hence, new forms of processing data are needed to enable enhanced decision making, insight, process optimization, data mining and knowledge discovery. This drives and motivates research and practices in Big data analytics [1], [23] and Big data mining [3],[20]. Having developed systematic or quantitative processes to mine and analyse Big data allows us to continuously or iteratively explore, investigate, and understand past business performance so as to gain new insight and drive science or business planning. To handle Big data, researchers proposed the use of a high-level programming model called MapReduce to process high volumes of data by using parallel and distributed computing [25] on large clusters or grids of nodes, commodity machines, which consist of a master node and multiple worker nodes. As implied by its name, MapReduce involves two key functions: "map" and "reduce". An advantage of using the MapReduce model is that users only need to focus on (and specify) these "map" and "reduce" functions without worrying about implementation details for (i) partitioning the input data, (ii) scheduling and executing the program across multiple machines, (iii) handling machine failures, or (iv) managing inter-machine communication.

2. BACKGROUND

Here, we provide some background information about (A) mining uncertain data, (B) mining with constraints, (C) the MapReduce model and (D) mining with the MapReduce model.

A. Mining Frequent Patterns from Uncertain Data

Let (i) Item be a set of m domain items and (ii) $X = \{x1, x2, \dots, x\ k\}$ 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 $tj \subseteq$ Item (for $1 \leq j \leq n$). The projected database of X is the set of all transactions containing X . Unlike precise databases, each item xi in a transaction $tj = \{x1, x2,\dots, xk\}$ in an uncertain database is associated with an **existential probability value** $P(xi, tj)$, which represents the likelihood of the presence of xi in tj [11]. Note that $0 < P(xi, tj) \leq 1$. The **existential probability** $P(X, tj)$ **of a pattern** X **in** tj

It is then the product of the corresponding existential probability values of every item *x* within *X* when these items are indepen- dent [11]: *P* (*X, tj*) = *x*∈*X P* (*x, tj*). The **expected support *expSup*(*X*)** of *X* in the database is the sum of *P* (*X, tj*) over all *n* transactions in the database

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

where *P*(*x, tj*) is the existential probability value of item *x* intransaction *tj*. With this notion of expected support, existing tree-based algorithms—such as UF-growth [14], CUF-growth [15] and PUF-growth [16]—mine frequent patterns from uncertain data as follows. The algorithms first scan the uncertain database once to compute the expected support of all domain items (i.e., singleton itemsets). 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 [14], CUF-tree [15], or PUF-tree [16]). Each node in the tree captures (i) an item *x*, (ii) its existential probability *P*(*x, tj*), and (iii) 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*) ≥ a user-specified minimum support threshold *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*.

B. Mining Frequent Patterns that Satisfy User-Specified Constraints

An existing constrained frequent pattern mining framework [9], [10], [19] allows the user to use a set of SQL-style constraints to specify his interest for guiding the *precise data* mining process so that only those frequently occurring sets of market basket items satisfying the user-specified constraints are found. This avoids unnecessary computation for mining those uninteresting frequent patterns. Besides market basket items, the set of constraints can also be imposed on items, events or objects in other domains. The following are some examples of user constraints. Constraint *C*1 ≡*m in*(*X.Snowfall*) ≥ 4cm expresses the user interest in finding every frequent pattern *X* such that the minimum amount of snowfall among all meteorological records in *X* is at least 4cm. Similarly, *C*2 ≡ *max*(*X.Temperature*) ≤ −10 ◦C says that the maximum temperature among all meteorological records matching a pattern *X* is at most −10 ◦C. Constraint *C*3 ≡ *X.Location* = *Anchorage* expresses the user interest in finding every frequent pattern *X* such that all vents in *X* are held in Anchorage, AK, USA; *C*4 ≡ *X.Weight* ≥ 32kg says that the weight of each object in *X* is at least 32kg. Constraints *C*5 ≡ *sum*(*X.Snowfall*) ≤ 15cm says that the cumulative snowfall on all selected meteorological records in *X* is at most 15cm; *C*6 ≡ *diff*(*X.Temperature*) = *max*(*X.Temperature*) − *min*(*X.Temperature*) ≤ 20 ◦C says that the difference between the maximum and minimum temperatures in *X* is at most 20 ◦C. User-specified constraints can generally be categorized into several overlapping classes according to the properties that they possess. The first four aforementioned constraints in particular can be categorized into a popular class of constraints called **succinct anti-monotone (SAM) constraints**, which possess the properties of both *succinctness* and *anti-monotonicity*.

Definition 1: An itemset *SSj* ⊆ I t e m is a *succinct set* if it can be expressed as a result of selection operation *σp* where (i) *σ* is the usual SQL-style selection operator, (ii) *p* is a selection predicate and (iii) Item is a set of domain items. A powerset of items *SP* ⊆ 2Item is a *succinct powerset* if there is a fixed number of succinct sets *SS*1, . . ., *SSk* ⊆ Item such that *SP* can be expressed in terms of the powersets of *SS*1, . . ., *SSk* using set union and/or set difference operators. A constraint *C* is **succinct** [10] provided that the collection of patterns satisfying *C* is a succinct powerset.

Definition 2: A constraint *C* is **anti-monotone** [10] if and only if all subsets of a pattern satisfying *C* also satisfy *C*. A frequent pattern *X* is **valid** in an uncertain database if such a frequent pattern also satisfies the user-specified constraints. Given (i) an uncertain database, (ii) *minsup* and (iii) user-specified constraints (e.g., the SAM constraints), the research problem of **constrained frequent pattern mining from uncertain data** is to discover from the database a complete set of patterns having expected support ≥ *minsup* (i.e., frequent patterns), which also satisfy the user-specified constraints (i.e., valid patterns).

C. The MapReduce Programming Model As a high-level programming model for processing vast amounts of data, *MapReduce* [6] usually uses parallel and distributed computing on clusters or grids of nodes (i.e., computers). The ideas behind MapReduce can be described as follows. As implied by its name, MapReduce involves two key functions: “map” and “reduce”. The input data are read, divided into several partitions (sub problems), and assigned to different processors. Each processor executes the **map function** on each partition (subproblems). The map function takes a pair of *key, value* data and returns a list of *key, value* pairs as an intermediate result: map: *key*1, *value*1 → list of *key*2, *value*2 , where (i) *key*1 & *key*2 are keys in the same or different domains, and (ii) *value*1 & *value*2 are the corresponding values in some domains. Afterwards, these pairs are shuffled and sorted. Each processor then executes the **reduce function** on (i) a single key from this intermediate result together with (ii) the list of all values that appear with this key in the intermediate result. The reduce function “reduces”—by combining, aggregating, summarizing, filtering, or transforming the list of values associated with a given key (for all *k* keys) and returns (i) a list of *k* pairs of keys and values, (ii) a list of *k* values, or simply (iii) a single (aggregated or summarized) value: reduce: *key*2, list of *value*2 →list of *key*3, *value*3 reduce: *key*2, list of *value*2 → list of *value*3, or reduce: *key*2, list of *value*2 → *value*3, where (i) *key*2 is a key in some domains, and (ii) *value*2 & *value*3 are the corresponding values in some domains. Examples of MapReduce applications include the construction of an inverted index as well as the word counting of a document.

D. Mining Frequent Patterns Using the MapReduce Programming Model

Earlier works on MapReduce focused either on data processing [6] or on some data mining tasks other than frequent pattern mining (e.g., outlier detection [7], structure mining [24]). Recently, Lin et al. [17] proposed three Apriori based algorithms called SPC, FPC and DPC to mine frequent patterns from *precise data*. Among them, SPC uses single-pass counting to find frequent patterns of cardinality *k* at the *k*-the pass (i.e., the *k*-th database scan) for $k \geq 1$. FPC uses fixed passes combined-counting to find all patterns of cardinalities $k, (k + 1), ..., (k + m)$ in the same pass or database scan. On the one hand, this fixed-passes technique fixes the number of required passes from *K* (where *K* is the maximum cardinality of all frequent patterns that can be mined from the precise data) to a user-specified constant. On the other hand, due to combined-counting, the number of generated candidates is higher than that of SPC. In contrast, DPC uses dynamic-passes combined-counting, which takes the benefits of both SPC and FPC by taking into account the workloads of nodes when mining frequent patterns with MapReduce. Like these three algorithms, our proposed algorithm also uses MapReduce. However, unlike these three algorithms (which mine frequent patterns from *precise data* using the *Apriori-based approach*), our proposed algorithm mines frequent patterns from *uncertain data* using a *tree-based approach*. Note that the search space for frequent pattern mining for uncertain data is much larger than that for precise data due to the presence of the existential probability values.

3. OUR ALGORITHM FOR MINING UNCERTAIN BIG DATA WITH MAPREDUCE

Given (i) uncertain Big data, (ii) user-defined minsup, (iii) a user-specified constraint C (e.g., a SAM constraint), the research problem of constrained frequent pattern mining from uncertain Big data is to discover from Big data a complete set of patterns having expected support ≥ minsup and satisfying C (i.e., valid frequent patterns).

In this section, we propose our algorithm—which uses MapReduce—to mine valid frequent patterns from high volumes of uncertain data in a tree-based pattern-growth fashion (i.e., in a divide-and-conquer fashion). The algorithm uses two sets of the “map” and “reduce” functions during the Big data mining process: (A) One set for mining frequent singletons and (B) another set for mining frequent non-singleton patterns.

A. Mining Valid Frequent Singletons from Big Data

The key idea behind how our proposed algorithm mines frequent patterns (both singletons and non-singletons) that satisfy SAM constraints is based on the following observations.

Observation 1 : Due to succinctness, we can precisely enumerate all and only those patterns that satisfy the SAM constraints by using a member generating function. For example, the set of patterns satisfying

$C1 \equiv \min(X.\text{Snowfall}) \geq 4\text{cm}$,

which expresses the user interest in finding every frequent pattern X such that the minimum amount of snowfall among all meteorological records in each X is at least 4cm, is a succinct powerset. Thus, the set of patterns satisfying C1 can be expressed as $2^{\sigma_{\text{Snowfall} \geq 4\text{cm}}(\text{Item})}$. The corresponding member generating function can be represented as $\{X \mid X \subseteq \sigma_{\text{Snowfall} \geq 4\text{cm}}(\text{Item})\}$, which precisely enumerates all and only those patterns that satisfy C1: All these patterns must be comprised of only records with snowfall ≥ 4cm. Consequently, valid frequent patterns for C1 would be those frequent ones among the valid patterns satisfying C1.

Observation 2 : Due to anti-monotonicity, if a pattern does not satisfy the SAM constraints, all its supersets are guaranteed not to satisfy the SAM constraints. Thus, any pattern that does not satisfy the SAM constraints can be pruned. With the above observations, our proposed algorithm mines frequent patterns that satisfy the user-specified SAM constraints by performing the following key steps. First, our algorithm reads high volumes of uncertain Big data. As each item in the uncertain Big data is associated with an existential probability value, the algorithm computes the expected support of all domain items (i.e., singleton patterns) by using MapReduce. The expected support of any pattern can be computed by using Equation (1). Moreover, when computing singleton patterns, such an equation can be simplified to become the following:

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

where P(x, tj) is an existential probability of item x in transaction tj. Specifically, the algorithm divides the uncertain Big data into several partitions and assigns them to different processors. The map function receives transaction ID, content of that transaction as input. For every transaction tj, the map function emits a key, value pair for each item x ∈ tj. The question is: What should be the key and value in the emitted pair? A native attempt is to emit x, 1 for each occurrence of x ∈ tj. It would work well when mining precise data because each occurrence of x leads to an actual support of 1. In other words, occurrence of x is the same as the actual support of x when mining precise data. However, this is not the case when mining uncertain data. The occurrence of x can be different from the expected support of x when mining uncertain data. For instance, consider an item d with existential probability of 0.9 that appears only once in the entire uncertain Big database. Its expected support may be higher than that of another item f, which appears three times but with an existential probability of 0.2 in each appearance. Then, expSup({d }) = 0.9 > 0.6 = expSup({f }).Hence, instead of emitting x, 1 for each occurrence of x ∈ t j, our algorithm emits x, P(x, tj) for each occurrence of x ∈ tj. In other words, the map function can be specified as follows:

```
For each tj ∈ partition of the uncertain Big data do
for each item x ∈ tj do
emit x, P(x, tj) .
```

This results in a list of x, P(x, tj) pairs with many different x and P(x, tj) for the keys and values. Afterwards, these x, P(x, tj) pairs are shuffled and sorted to form x, list of

P(x, tj) . Each processor then executes the reduce function on these shuffled and sorted x, list of P(x, tj) pairs and applies constraint checks on every item x to obtain the expected support of only valid x (i.e., {x } that satisfies the SAM constraint CSAM). In other words, the reduce function can be specified as follows:

```
For each x ∈ x, list of P(x, tj) do
if {x } satisfies CSAM then
set expSup( {x }) = 0;
for each P(x, tj) ∈ list of P(x, tj) do
expSup( {x }) = expSup( {x }) + P(x, tj);
if expSup( {x }) ≥ minsup then
emit {x }, expSup( {x }) .
```

In a high-level abstract view, the set of the map and reduce functions for mining valid frequent singletons from Big data can be described as follows: map: ID of transaction tj, content of tj → list of x, P(x, tj) , in which the master node reads and divides uncertain Big data in partitions. The worker node corresponding to each partition then outputs the x, P(x, tj) pairs for each domain item x. Then, the reduce function sums all existential probabilities of x for each valid x to compute its expected support: reduce: x, list of P(x, tj) →list of valid frequent {x }, expSup({x }) .

Example 1: Let us consider a tiny sample set of an uncertain Big database and its auxiliary information as shown
in Table I with (i) the user-defined minsup=0.9 and (ii) a user-specified constraint CSAM ≡ min(X.Snowfall) ≥ 4cm (which expresses the user interest in finding every frequent pattern X such that the minimum amount of snowfall among all meteorological records in X is at least
4cm). Based on the auxiliary information, we learn that domain items a, b, c, e & f (but not d) satisfy CSAM.
Then, for the first transaction t1, the map function outputs a, 0.9 , b, 1.0 , c, 0.5 , d, 0.9 , e, 1.0 , f, 0.2 . Similarly,
for the second transaction t2, the map function outputs a, 0.8 , b, 0.8 , c, 1.0 , e, 0.2 , f, 0.2 ; for the third transaction t3, the map function outputs a, 0.4 , f, 0.2 . These pairs are then shuffled and sorted. Afterwards, the reduce Function is read as, [0.9, 0.8, 0.4] , b, [1.0, 0.8] , c, [0.5, 1.0] , d, [0.9] , e, [1.0, 0.2] & f, [0.2, 0.2, 0.2] , and outputs {a }, 2.1 , {b }, 1.8 , {c }, 1.5 & {e }, 1.2 (i.e., valid singletons and their corresponding expected support). Note that the reduce function does not read d, [0.9], let alone output {d}, 0.9, because {d} does not satisfy CSAM. On the other hand, although the reduce function reads f, [0.2, 0.2, 0.2] It does not output {f}, 0.6 because valid {f} is infrequent.

B . Min in g Valid Frequent Non - singleton Patterns from Big Data After applying the first set of map-reduce functions, we obtain all valid frequent singletons (i.e., domain items that satisfy the user-specified constraints) and their associated existential support values. The next step is an important and computationally intensive step. Our proposed algorithm rereads each transaction in the uncertain Big database to form an {x}-projected database (i.e., a collection of transactions containing x) for each valid frequent singleton {x} returned by the first reduce function. The map function can be specified as follows:

```
For each tj ∈ partition of the uncertain Big data do
for each {x } ∈ {x }, expSup( {x }) do
emit {x }, prefix of tj ending with x .
```

As all valid patterns must be comprised of only valid singleton
items (due to the succinctness and anti-monotonicity), our proposed algorithm keeps only those valid singleton items returned by the first reduce function in each prefix of tj when forming an {x }-projected database. Note that no additional constraint check is required when forming the projected database or mining frequent patterns.
The worker node corresponding to each projected database then builds appropriate trees (e.g., UF-tree, CUF-tree, or PUF-tree) based on the projected databases assigned to the worker node to mine every valid frequent non-singleton pattern X (with cardinality k, where k ≥ 2). The worker node also outputs X, expSup(X), i.e., every valid frequent non-singleton pattern with its expected support:

```
For each {x } ∈ {x }-projected database do
build a tree for the {x }-projected database to find X;
if expSup(X) ≥ minsup then
emit X, expSup(X)
```

In a high-level abstract view, this second set of map-reduce functions for mining valid frequent non-singleton patterns from Big data can be described as follows: map: ID of transaction tj, content of tj → list of valid frequent {x }, part of tj with x , (5) in which the master node rereads and divides uncertain Big data in partitions. The worker node corresponding to each partition helps to form an {x }-projected database for every valid frequent item x in the transactions assigned to that partition. The {x}-projected database consists of prefixes of relevant transactions (from the uncertain Big database) that end with x. More precisely, the worker node outputs {x}, portion of tj for forming the {x}-projected database pairs. Then, the reduce function shuffles and sorts these pairs of {x }- projected databases, from which valid frequent non-singleton patterns can be found and their expected support values can be computed: reduce: valid frequent {x }, {x }-projected database→ list of valid frequent X, expSup(X) . (6)

Example 2: Let us continue with Example 1, where
(i) minsup=0.9 and (ii) CSAM ≡ min(X.Snowfall) ≥ 4cm.
Recall that {a}, {b}, {c} and {e} are valid frequent singletons. Our algorithm rereads the uncertain Big database. After reading the first transaction t1, the (second) map function outputs {b }, {a:0.9, b:1.0 } (where {a:0.9, b:1.0 } is a prefix of t1 ending with item b), {c }, {a:0.9, b:1.0, c:0.5 } and {e }, {a:0.9, b:1.0, c:0.5, e:1.0 } (where {a:0.9, b:1.0, c:0.5, e:1.0 } contains only valid frequent items—i.e., it does not contain invalid item d). Note that this map function does not output {a}, {a: 0.9} because {a: 0.9 } does not contain any valid frequent item other than itself (i.e., such a prefix of t1 does not contribute to the mining of non-singletons). Moreover, this map function does not output{f }, a:0.9, b:1.0, c:0.5,e:1.0, f:0.2 } either, but due to a different reason. The reason here is because {f} is invalid. Similarly, after reading the second transaction t2, the map function outputs {b }, {a:0.8,b:0.8 } ,

{c }, {a:0.8, b:0.8, c:1.0 } and {e }, {a:0.8, b:0.8,c:1.0, e:0.2 } . These pairs are then shuffled and sorted. Afterwards, the reduce function reads {b}, {b}-projected database. Based on this {b}-projected database (which consists of two sub transactions {a: 0.9, b: 1.0} and {a: 0.8, b: 0.8}), a tree is built. Consequently, valid frequent pattern {a, b} with an expected support of 1.54 is found. Similarly, the reduce function reads {c}, {c}-projected database. It builds a tree based on this {c }-projected database (which consists of two sub transactions {a:0.9, b:1.0, c:0.5 } and {a:0.8, b:0.8, c:1.0 }), and finds valid frequent patterns {a, c }, {a, b, c } & {b, c } with expected support values of 1.25, 1.09 & 1.3, respectively. The reduce function then reads {e}, {e}-projected database. It builds a tree based on this {e }-projected database (which consists of two sub transactions {a:0.9, b:1.0, c:0.5, e:1.0 } and {a:0.8, b:0.8, c:1.0, e:0.2 }), and finds valid frequent patterns{a, e } & {b,e} with expected support values of 1.06 & 1.16.

To summarize, the first set of map-reduce functions discover four valid frequent singletons (with their corresponding expected support values): {a}, 2.1, {b}, 1.8, {c}, 1.5 and {e}, 1.2. The second set of map-reduce functions discover six valid frequent non-singleton patterns (with their corresponding expected support values): {a, b }, 1.54 , {a, b, c }, 1.09 , {a, c }, 1.25 , {a, e }, 1.06 ,{b, c }, 1.3 and {b, e }, 1.16 . Hence, our algorithm finds a total of ten frequent patterns satisfying CSAM.

C. Pushing Constraint Checks into the First Map Function

Recall from Section III-A that the first reduce function applies constraint checks to verify whether {x } satisfies the user-specified CSAM (i.e., if {x } is valid), then the reduce function computes the expected support of valid singleton x and returns only frequent singleton {x }. Alternatively, to handle the user-specified SAM constraint CSAM, we could push CSAM into the mining process earlier (by pushing it into the map function instead of the reduce function as shown in Section III-A). For instance, we could push the constraint checking into the map function so that we only emit x, P(x, tj) for each item x ∈ tj that satisfy CSAM:

```
For each tj ∈ partition of the uncertain Big data do
For each item x ∈ tj and {x} satisfies CSAM do
emit x, P(x, tj) .
```

Afterwards, these valid x, P(x, tj) pairs are shuffled and sorted. Each processor then executes the reduce function on the shuffled and sorted pairs to obtain the expected support of x. In other words, an alternative reduce function can be specified as follows:

```
For each x ∈ valid x, list of P(x, tj) do
set expSup( {x }) = 0;
for each P(x, tj) ∈ list of P(x, tj) do
expSup( {x }) = expSup( {x }) + P(x, tj);
if expSup( {x }) ≥ minsup then
emit {x}, expSup( {x}) .
```

In a high-level abstract view, this alternative set of the map and reduce functions for mining valid frequent singletons from Big data can be described as follows:

map: ID of transaction tj, content of tj → list of valid x, P(x, tj) , (7) in which the master node reads and divides uncertain Big data in partitions. The worker node corresponding to each partition then outputs the valid x, P(x, tj) pairs for each domain item x. Then, the reduce function sums all existential probabilities of x for each valid x to compute its expected support: reduce: valid x, list of P(x, tj) →list of valid frequent {x}, expSup ({x}). (8)

Example 3 : For comparison between the two approaches for mining valid frequent singletons from Big data (i.e., comparing the current approach of pushing constraint checks into the first map function with the approach of pushing constraint checks into the first reduce function), let us revisit

Example 1. We mine the same a tiny sample set of an uncertain Big database as shown in Table I with (i) min su p=0.9 and (ii) CSAM ≡ min(X.Snowfall) ≥ 4cm. Again, based on the auxiliary information, we learn that domain items a, b, c, e & f (but not d) satisfy CSAM. Then, for the first transaction t1, the map function outputs a, 0.9 , b, 1.0 , c, 0.5 , e, 1.0 , f, 0.2 . Note that, by pushing the constraint checks into the map function, we no longer emit d, 0.9 —let alone perform any bookkeeping in the reduce function—because {d } does not satisfy CSAM. Similarly, for the second transaction t2, the map function outputs a, 0.8 , b, 0.8 , c, 1.0 , e, 0.2 , f, 0.2 ; for the third transaction t3, the map function outputs a, 0.4 , f, 0.2 . These pairs are then shuffled and sorted. Afterwards, the reduce function reads a, [0.9, 0.8, 0.4] , b, [1.0, 0.8] , c, [0.5, 1.0] , e, [1.0, 0.2] & f, [0.2, 0.2, 0.2] , and outputs {a }, 2.1 , {b }, 1.8 , {c }, 1.5 & {e }, 1.2 as valid singletons and their corresponding expected support values. Although the reduce function reads f, [0.2, 0.2, 0.2] , it does not output {f}, 0.6 because valid {f } is infrequent. Note that, as the map function does not emit d, [0.9] , the reduce function does not need to read the invalid {d} and thus saves some computation. As observed from the above example, there are pros and cons between the two approaches for mining valid frequent singletons from Big data. For instance, the approach that pushes constraint checks into the first reduce function (as presented in Section III-A) requires fewer constraint checks because the constraint checks are delayed until all pairs are shuffled and sorted. Consequently, it only performs constraint checks on at most m domain items to see if they satisfy CSAM. In contrast, the approach that pushes constraint checks into the first map function (as presented here in this section) performs constraint checks on all occurrences of items in every transaction in the Big uncertain database, which is normally. Hence, pushing constraint checks into the reduce function is time-efficient (due to the reduction in the number of constraint checks), especially when the uncertain Big data consist of only a few domain items such as DNA sequences (in which the number of domain items are m=4 nucleobases “A”, “T”, “C” & “G”) or RNA sequences (in which the number of domain items are m=4 nucleobases “A”, “U”, “C” & “G”). On the other hand, the approach that pushes constraint checks into the first map function (as presented here in this section) requires less bookkeeping because it emits x, P(x, tj) pairs only for those items that satisfy CSAM. Consequently, fewer pairs need to be shuffled and sorted. Hence, pushing constraint checks into the map function is both space-efficient (due to the reduction in the number of pairs returned by the map function) and time-efficient (due to the reduction in the number of pairs to be shuffled and sorted), especially when high volumes of high-variety data come at a high velocity such as Big data streams.

D. Mining Frequent Patterns that Satisfy AM Constraints

So far, we have described how our proposed algorithm mines uncertain Big data with MapReduce for frequent patterns that satisfy user-specified SAM constraints. Although many of the commonly used constraints (e.g., the first four constraints C1–C4 mentioned in Section II-B) possess the properties of both anti-monotonicity and succinctness, there are some constraints that do not possess both properties (e.g., non-SAM constraints C5 & C6 in Section II-B). To handle these non-SAM constraints, we exploit one of these properties.

For instance, we exploit the property of anti-monotonicity. Based on Definition 2, if a pattern X does not satisfy an anti-monotone (AM) constraint CAM, then all its supersets do not satisfy CAM. Hence, such an invalid X can be pruned as it does not contribute to the mining of frequent patterns of future (higher) cardinalities. In this case, we adapt our algorithm by exploiting the property of anti-monotonicity as follows. We reuse the same approach as described in Section III-A or III-C to mine valid frequent singletons from an uncertain Big database using the first set of map-reduce functions. At the end of this step, the reduce function returns a list of valid frequent singletons with their expected support values. However, the second set of map-reduce functions for handling AM constraints is different due to the following observations.

Observation 3: Let V be the set of domain items that individually satisfy the SAM constraints. Then, due to succinctness and anti-monotonicity, any non-singleton pattern X comprising items from V (i.e., X ⊆ V) is guaranteed to satisfy the same SAM constraints. Thus, no constraint checks need to be performed when mining valid non-single to n patterns.

Observation 4: Let V be the set of domain items that individually satisfy the AM (but not succinct) constraints. Then, due to anti-monotonicity, n o t every non-singleton pattern X comprising items from V is guaranteed to satisfy the same AM constraints. In other words, X ⊆ V may be invalid, and thus constraint checks need to be performed when mining valid non-singleton patterns. Based on Observation 4, we adapt our algorithm by performing additional constraint checks in the second reduce function for handling AM constraints (cf. the second reduce function for handling SAM constraints described in Section III-B):

```
For each {x} ∈ {x}-projected database do
build a tree for the {x }-projected database to find X;
if expSup (X) ≥ minsup and X satisfies CAM then
emit X, expSup(X) .
```

4. EXPERIMENTAL RESULTS

In this section, we evaluate our proposed algorithm in mining user-specified constraints from uncertain Big data. We used different benchmark datasets, which include real-life datasets (e.g., accidents, connect4, and mushroom) from the UCI Machine Learning Repository available in (http://archive.ics.uci.edu/ml/) and the FIMI Repository (http://fimi.ua.ac.be/). We also used IBM synthetic datasets, which were generated using the IBM Quest Dataset Generator [2]. For our experiments, the generated data ranges from 2M to 10M transactions with an average transaction length of 10 items from a domain of 1K items. As the above real-life and synthetic datasets originally contained only precise data, we assigned to each item contained in every transaction an existential probability from the range (0,1]. All experiments were run using either (i) a single machine with an Intel Core i3 2-core processor (1.73 GHz) and 8 GB of main memory running a 64-bit Windows 7 operating system, or (ii) the Amazon Elastic Compute Cloud (EC2) cluster specifically, High-Memory Extra Large (m2.xlarge) computing nodes (http://aws.amazon.com/ec2).Experimental results show that, in terms of accuracy, our algorithm returned the same collection of valid frequent patterns as those returned by the existing mining framework [9], [10], [19] for finding valid frequent patterns from precise data. However, in terms of flexibility, our algorithm is not confined to finding valid frequent patterns from a database in which existential probability values of all items are 1. Our algorithm is capable of finding valid frequent patterns from any database, in which existential probability values of all items are ranging from 0 to 1. The same comments apply to the experimental results for our similar experiment with a user-specified AM constraint. Moreover, we also experimented with (i) an uncertain database and (ii) a user specified SAM constraint with 100% selectivity (so that every item is selected). Experimental results show that, in terms of accuracy, our algorithm returned the same collection of frequent patterns as those returned by the UF-growth [14], CUF-growth [15] and PUF-growth [16]. However, in terms of flexibility, our algorithm is not confined to handling SAM constraints with 100% selectivity. Our algorithm is capable of handling SAM constraints with any selectivity. Again, the same comments apply to the experimental results for our similar experiment with a user-specified AM constraint.

In the second experiment, we demonstrated the efficiency of our algorithm. (e.g., < 20,000 seconds) than the runtimes (e.g., > 120,000 seconds) required by UF-growth [14]. Moreover, shows that our algorithm led to high speedup (e.g., 7 to 10 times) even with just 11 nodes. Furthermore, show the runtimes of our algorithm decreased when the user defined minsup increased. In the third experiment, we demonstrated the benefits of constraint pushing in reducing the search space of Big data mining. The benefits become more obvious they show that, when selectivity decreased (i.e., fewer frequent patterns satisfy the constraints), runtimes also decreased, because (I) fewer pairs were returned by the map function, (ii) fewer pairs were shuffled and sorted by the reduce function, and/or (iii) fewer constraint checks were performed.

5. CONCLUSIONS

Existing algorithms mostly focus on association analysis enabled by mining interesting patterns from precise databases. However, there are situations in which data are uncertain. As items in each transaction of these probabilistic databases of uncertain data are usually associated with existential probabilities expressing the likelihood of these items to be present in the transaction, the search space for mining from uncertain data is much larger than mining from precise data. This matter is worsened as we move into the era of Big data. Furthermore, in many real-life applications, users may be interested in only a tiny portion of this large search space. To avoid wasting lots of time and space in computing all frequent patterns first and pruning uninteresting ones as a post-processing step, we proposed in this paper a tree-based algorithm that (I) allows users to express their interest in terms of succinct anti-monotone (SAM) constraints and (ii) uses MapReduce to mine uncertain Big data for frequent patterns that satisfy the user-specified constraints. As a result, our algorithm returns all and only those patterns that are interesting to the users. Moreover, although we focused mostly on handling SAM constraints, we also discussed how our algorithm handles constraints that are anti monotone (AM) but not succinct.

REFERENCES

[1] P. Agarwal, G. Shroff, & P. Malhotra, "Approximate incremental big- data harmonization," in IEEE Big Data Congress 2013, pp. 118–125.

[2] R. Agrawal & R. Srikant, "Fast algorithms for mining association rules," in VLDB 1994, pp. 487–499.

[3] A. Azzini & P. Ceravolo, "Consistent process mining over Big data triple stores," in IEEE Big Data Congress 2013, pp. 54–61.

[4] T. Condie, P. Mineiro, N. Polyzotis, & M. Weimer, "Machine learning for Big data," in ACM SIGMOD 2013, pp. 939–942.

[5] R.L.F. Cordeiro, C. Traina Jr., A.J.M. Traina, J. Lo̍pez, U. Kang, & C. Faloutsos, "Clustering very large multi-dimensional datasets with MapReduce," in ACM KDD 2011, pp. 690–698.

[6] J. Dean & S. Ghemawat, "MapReduce: simplified data processing on large clusters," CACM 51(1): 107–113, Jan. 2008.

[7] A. Koufakou, J. Secretan, J. Reeder, K. Cardona, & M. Georgiopoulos, "Fast parallel outlier detection for categorical datasets using MapRe- duce," in IEEE IJCNN 2008, pp. 3298–3304.

[8] A. Kumar, F. Niu, & C. Ré, "Hazy: making it easier to build and maintain Big-data analytics," CACM 56(3): 40–49, Mar. 2013.

[9] L.V.S. Lakshmanan, C.K.-S. Leung, & R.T. Ng, "Efficient dynamic mining of constrained frequent sets," ACM TODS 28(4): 337–389, Dec. 2003.

[10] C.K.-S. Leung, "Frequent itemset mining with constraints," in Encyclo- pedia of Database Systems, pp. 1179–1183, 2009.

[11] C.K.-S. Leung, "Mining uncertain data," WIREs Data Mining and Knowledge Discovery 1(4): 316–329, July/Aug. 2011.

[12] C.K.-S. Leung & F. Jiang, "Frequent itemset mining of uncertain data streams using the damped window model," in ACM SAC 2011, pp. 950– 955.

[13] C.K.-S. Leung & F. Jiang, "Frequent pattern mining from time-fading streams of uncertain data," in DaWaK 2011 (LNCS 6862), pp. 252–264.

[14] C.K.-S. Leung, M.A.F. Mateo, & D.A. Brajczuk, "A tree-based ap- proach for frequent pattern mining from uncertain data," in PAKDD 2008 (LNAI 5012), pp. 653–661.

[15] C.K.-S. Leung & S.K. Tanbeer, "Fast tree-based mining of frequent itemsets from uncertain data," in DASFAA 2012 (LNCS 7238), pp. 272– 287.

[16] C.K.-S. Leung & S.K. Tanbeer, "PUF-tree: A compact tree structure for frequent pattern mining of uncertain data," in PAKDD 2013 (LNCS 7818), pp. 13–25.

[17] M.-Y. Lin, P.-Y. Lee, & S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," in ICUIMC 2012, art. 76.

[18] S. Madden, "From databases to big data," IEEE Internet Computing, 16(3): 4–6, May–June 2012.

[19] R.T., Ng, L.V.S. Lakshmanan, J. Han, & A. Pang, "Exploratory mining and pruning optimizations of constrained associations rules," in ACM SIGMOD 1998, pp. 13–24.

[20] E. Ö lmezoğullari & I. Ari, "Online association rule mining over fast data," in IEEE Big Data Congress 2013, pp. 110–117.

[21] M. Riondato, J. DeBrabant, R. Fonseca, & E. Upfal, "PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce," in ACM CIKM 2012, pp. 85–94.

[22] Y. Tong, L. Chen, Y. Cheng, & P.S. Yu, "Mining frequent itemsets over uncertain databases," PVLDB, 5(11): 1650–1661, July 2012.

[23] H. Yang & S. Fong, "Countering the concept-drift problem in big data using iOVFDT," in IEEE Big Data Congress 2013, pp. 126–132.

[24] S. Yang, B. Wang, H. Zhao, & B. Wu, "Effi dense structure mining using MapReduce," in IEEE ICDM Workshops 2009, pp. 332–337.

[25] M.J. Zaki, "Parallel and distributed association mining: a survey," IEEE Concurrency, 7(4): 14–25, Oct.–Dec. 1999.