BigSAM: Mining Interesting Patterns from Probabilistic Databases of Uncertain Big Data

Fan Jiang, Carson Kai-Sang Leung(✉), and Richard Kyle MacKinnon

University of Manitoba, Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

Abstract. Nowadays, high volumes of valuable uncertain data can be easily collected or generated at high velocity in many real-life applications. Mining these uncertain Big data is computationally intensive due to the presence of existential probability values associated with items in every transaction in the uncertain data. Each existential probability value expresses the likelihood of that item to be present in a particular transaction in the Big data. In some situations, users may be interested in mining all frequent patterns from these uncertain Big data; in other situations, users may be interested in only a tiny portion of these mined patterns. To reduce the computation and to focus the mining for the latter situations, we propose a tree-based algorithm that (i) allows users to express the patterns to be mined according to their intention via the use of constraints and (ii) uses MapReduce to mine uncertain Big data for only those frequent patterns that satisfy user-specified constraints. Experimental results show the effectiveness of our algorithm in mining probabilistic databases of uncertain Big data.

1 Introduction and Related Works

Nowadays, high volumes of valuable data are easily collected or generated in various real-life application areas such as bioinformatics, e-commerce, healthcare, mobile sensing, security, and social networks. This leads us into the new era of Big data [20]. Intuitively, *Big data* refer to high-velocity, high-value, and/or high-variety data with volumes beyond the ability of commonly-used software to capture, curate, manage, and process within a tolerable elapsed time. Hence, new forms of processing data are needed to deliver high veracity (and low vulnerability) and to enable enhanced decision making, insight, knowledge discovery, and process optimization. This drives and motivates research and practices in business analytics and optimization, which require techniques like Big data mining and analytics [12,14]. Having developed systematic or quantitative processes to mine and analyze Big data allows us to continuously or iteratively explore, investigate, and understand the past business performance so as to gain new insight and drive 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

W.-C. Peng et al. (Eds.): PAKDD 2014 Workshops, LNAI 8643, pp. 780–792, 2014.
DOI: 10.1007/978-3-319-13186-3_70

distributed computing [25] on large clusters or grids of nodes (i.e., 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. Over the past few years, several algorithms have been proposed to use the MapReduce model—which applies distributed or parallel computing—for different Big data mining and analytics tasks [5,10]. Examples of these tasks include clustering [6], outlier detection [9], and structure mining [24]. An equivalently important mining and analytics task is *pattern mining* [3,18,23], which aims to analyze valuable data for the discovery of implicit, previously unknown, and potentially useful knowledge in the form of patterns. For example, *frequent patterns* help reveal collections of popular merchandise items, frequently co-occurring objects, or frequently co-located events.

Since the introduction of pattern mining [1], numerous algorithms have been proposed to mine *precise* data such as shopper market basket transaction databases. With these databases, users definitely know whether an item is present in—or is absent from—a transaction. In this notion, each item in a transaction t_j in databases of precise data can be viewed as an item with a 100 % likelihood of being present in t_j. However, data in many real-life applications are riddled with uncertainty [2,16,19]. It is partially due to inherent measurement inaccuracies, sampling and duration errors, network latencies, and intentional blurring of data to preserve anonymity. Hence, users are usually uncertain about the presence or absence of items. As a concrete example, a meteorologist may suspect (but cannot guarantee) that severe weather phenomena will develop during a thunderstorm. The uncertainty of such suspicions can be expressed in terms of existential probability. For instance, a thunderstorm may have a 60 % likelihood of generating hail, and only a 15 % likelihood of generating a tornado, regardless of whether or not there is hail. To handle uncertain data, several pattern mining algorithms—such as the U-Apriori [4], UF-growth [15], and PUF-growth [16] algorithms—were proposed in previous PAKDD conferences.

Furthermore, in many real-life applications, users may have some particular phenomena in mind on which to focus the mining (e.g., a meteorologist may want to find only those weather records about thunderstorms with hail). However, the aforementioned algorithms mine patterns without user focus. Consequently, users often need to wait for a long period of time for numerous patterns, out of which only a tiny fraction may be interesting to the users. Hence, *constrained pattern mining* [11]—which aims to find *valid* patterns (i.e., patterns that satisfy the user-defined constraints)—is needed.

A natural question to ask is: Can we use MapReduce to perform constrained pattern mining from uncertain Big data? In response to this question, we propose an algorithm—called BigSAM—to mine uncertain Big data for frequent patterns that satisfy a particular type of user-specified constraints called succinct anti-monotone (SAM) constraints. Our algorithm discovers patterns from probabilistic databases of uncertain Big data in a pattern-growth fashion for Big data analytics. Such an algorithm—which is a non-trivial integration of (i) mining for patterns, (ii) mining from uncertain data, (iii) mining from Big data, and (iv) mining with constraints—is our key contribution of this paper.

The remainder of this paper is organized as follows. The next section gives background about uncertain data, SAM constraints, and the MapReduce model. In Sect. 3, we propose our BigSAM algorithm for mining uncertain Big data for patterns that satisfy the user-specified SAM constraints using MapReduce. Evaluation results and conclusions are presented in Sects. 4 and 5, respectively.

2 Background

In this section, we give some background information about (i) uncertain data, (ii) SAM constraints, and (iii) the MapReduce model.

2.1 Uncertain Data: Existential Probability and Expected Support

Let (i) `Item` be a set of m domain items and (ii) $X = \{x_1, x_2, \ldots, x_k\}$ be a pattern comprising k items (i.e., a *k-itemset*), where $X \subseteq$ `Item` and $1 \leq k \leq m$. Then, each item x_i in a transaction $t_j = \{x_1, x_2, \ldots, x_h\} \subseteq$ `Item` in a probabilistic database of uncertain data is associated with an *existential probability value* $P(x_i, t_j)$ [13], which represents the likelihood of the presence of x_i in t_j. Note that $0 < P(x_i, t_j) \leq 1$. The existential probability $P(X, t_j)$ of a pattern X in t_j is the product of the corresponding existential probability values of every item x within X (when these items are independent) [13]: $P(X, t_j) = \prod_{x \in X} P(x, t_j)$.

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

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

where $P(x, t_j)$ is the existential probability value of item x in transaction t_j. With this definition of expected support, existing tree-based algorithms [8] such as UF-growth [15] and PUF-growth [16] mine frequent patterns from a probabilistic database of uncertain data as follows. The algorithms first scan the 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 (i.e., UF-tree [15] or PUF-tree [16]). Each node in the tree captures (i) an item x, (ii) its existential probability value $P(x, t_j)$, and (iii) its occurrence count. At each step during the mining process, the frequent patterns are expanded recursively.

Given such a database and a user-specified minimum support threshold *minsup*, the research problem of *frequent pattern mining from uncertain data* is to discover from the probabilistic database of uncertain data all those *frequent* patterns (i.e., patterns having expected support $\geq$ *minsup*).

2.2 Succinct Anti-monotone (SAM) Constraints

To mine interesting patterns from precise data, the Constrained Apriori (CAP) framework [21] allows the user to specify his interest via the use of SQL-style constraints to guide the mining process so that only those frequently occurring sets of shopper basket items satisfying the user constraints are returned. This avoids unnecessary computation for mining those uninteresting frequent patterns. Examples of user-specified constraints include (i) $max(\mathrm{X}.Price) \leq \$500$, which expresses the user's interest in finding every frequent pattern X such that the maximum price of all shopper market basket items in X is at most \$500, and (ii) $X.Type \neq snack$, which expresses the user's interest in finding every frequent pattern X such that every shopper market basket item in X is not a snack item.

In general, user-specified constraints can be categorized into several overlapping classes according to the properties that they possess. The two 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 (Succinctness [11]). Let `Item` be the set of m domain items. Then, an itemset $SS_j \subseteq \texttt{Item}$ is a *succinct set* if SS_j can be expressed as a result of selection operation σ_p (`Item`), where σ is the usual SQL-style selection operator and p is a selection predicate. A powerset of items $SP \subseteq 2^{\texttt{Item}}$ is a *succinct powerset* if there is a fixed number of succinct sets $SS_1, \ldots, SS_k \subseteq \texttt{Item}$ such that SP can be expressed in terms of the powersets of $SS_1, \ldots, SS_k$ using set union and/or set difference operators. A constraint C is ***succinct*** provided that the set of patterns satisfying C is a succinct powerset. □

Definition 2 (Anti-monotonicity [11]). A constraint C is ***anti-monotone*** if and only if all subsets of a pattern satisfying C also satisfy C. □

2.3 The MapReduce Programming Model

MapReduce [7] is a high-level programming model for processing vast amounts of data. Usually, MapReduce 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 (subproblems), and assigned to different processors. Each processor executes the *map function* on each partition (subproblem). The map function takes a pair of $\langle key, value\rangle$ data and returns a list of $\langle key, value\rangle$ pairs as an intermediate result:

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

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 value 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 a list of k values:

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

or returns a single (aggregated or summarized) value:

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

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.

Early works on MapReduce focused either on data processing [7] or on some data mining tasks other than frequent pattern mining (e.g., outlier detection [9], structure mining [24]). Recently, three Apriori-based algorithms called SPC, FPC and DPC [17] were proposed to mine frequent patterns from *precise data*. Like these three algorithms, our proposed BigSAM algorithm also uses MapReduce. However, unlike these three algorithms (which mine frequent patterns from *precise data* using the *Apriori-based approach*), our proposed BigSAM mines frequent patterns from *uncertain data* using a *tree-based approach*. The search/solution space for pattern mining from uncertain data is much larger than that for pattern mining from precise data due to the presence of the existential probabilities.

Moreover, a parallel randomized algorithm called PARMA [22] was proposed to mine *approximations* to the top-k frequent patterns and association rules from *precise data* using MapReduce. Although PARMA and our BigSAM algorithm both use MapReduce, one key difference between the two algorithms is that we aim to mine for *truly frequent* (instead of approximately frequent) patterns. Another key difference is that we mine from *uncertain data* (instead of precise data). The third key difference is that we mine with constraints so that we focus our computation on finding only those frequent patterns *that satisfy the use-specified SAM constraints* (instead of all frequent patterns).

3 Our BigSAM Algorithm that Mines Uncertain Big Data for Frequent Patterns Satisfying SAM Constraints

Given (i) a probabilistic database of uncertain Big data, (ii) a user-specified SAM constraint, and (iii) a user-specified minimum support threshold *minsup*, our proposed BigSAM algorithm uses the MapReduce programming model to mine uncertain Big data—in a tree-based pattern-growth fashion—for all patterns satisfying the SAM constraint and having expected support $\geq$ *minsup* (i.e., *valid frequent patterns*).

Recall from Sect. 2.2 that users can express their interests by specifying constraints. Our BigSAM algorithm does not confine the user-specified constraints to only shopper market basket items. We allow users to specify constraints that can be imposed on items, events, or objects in other domains. For example, constraint $C_1 \equiv max(X.Temperature) \leq 20°\text{C}$ expresses the meteorologist's interest in finding every frequent pattern X such that the maximum temperature of all meteorological records in a pattern X is at most 20°C. In other domain (e.g., healthcare sector), constraint $C_2 \equiv min(X.WBC) \geq 4.3 \times 10^9/\text{L}$ expresses the physician's interest in finding every group X such that the minimum white blood cell (WBC) counts among all patients in X is at least $4.3 \times 10^9/\text{L}$. For mobile sensing, constraint $C_3 \equiv X.Location{=}Tainan$ expresses the user interest in finding every frequent pattern X such that all events in X are held in the city of Tainan; constraint $C_4 \equiv X.Location{=}(Tainan \vee Kaohsiung)$ expresses the user interest in finding every frequent pattern X such that all events in X are held in Tainan or Kaohsiung. Constraint $C_5 \equiv X.Location{\neq}Winnipeg$ expresses the user interest in finding every frequent pattern X such that all events in X are held outside Winnipeg.

We observed that, 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. Moreover, due to succinctness, we can precisely enumerate all and only those patterns that satisfy the constraints by using a member generating function. For example, the set of patterns satisfying $C_1 \equiv max(X.Temperature) \leq 20°\text{C}$ is a succinct powerset. Thus, the set of patterns satisfying C_1 can be expressed as $2^{\sigma_{Temperature \leq 20°\text{C}}(\texttt{Item})}$. The corresponding member generating function can be represented as $\{X \mid X \subseteq \sigma_{Temperature \leq 20°\text{C}}(\texttt{Item})\}$, which precisely enumerates all and only those *valid* patterns (i.e., patterns that satisfy C_1): All these patterns must comprised only items with individual temperature $\leq 20°\text{C}$. Consequently, valid frequent patterns for C_1 would be those frequent ones among the valid patterns satisfying C_1.

3.1 Exploiting SAM Constraints in the Reduce Function

With the above observations, our BigSAM algorithm exploits the properties of succinctness and anti-monotonicity. The key idea of our algorithm—which uses two sets of the map and reduce functions to mine uncertain Big data for frequent patterns satisfying SAM constraints—can be described as follows. BigSAM first reads high volumes of uncertain Big data. As each item in the volumes is associated with an existential probability, BigSAM aims to compute 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 Eq. (1). Moreover, for any singleton pattern $\{x\}$, such an equation can be simplified to become the following:

$$expSup(\{x\}) = \sum_{j=1}^{n} P(x, t_j), \tag{2}$$

where $P(x, t_j)$ is the existential probability of item x in transaction t_j. Specifically, BigSAM divides the uncertain 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 t_j, the map function emits an $\langle x, P(x, t_j)\rangle$ pair for each item $x \in t_j$ (cf. when mining *precise* data, each occurrence leads to an actual support of 1 and would yield a corresponding $\langle x, 1\rangle$ pair). When mining *uncertain* data, the occurrence of x can be different from the expected support of x. For instance, consider an item a with existential probability value of 0.9 that appears only in transaction t_1. Its expected support may be higher than item b that appears seven times but with an existential probability value of 0.1 in each appearance. Then, $expSup(\{a\}) = 0.9 > 0.7 = expSup(\{b\})$. Hence, instead of emitting $\langle x, 1\rangle$ for each occurrence of $x \in t_j$, BigSAM emits $\langle x, P(x, t_j)\rangle$ for each occurrence of $x \in t_j$. In other words, the map function can be specified as follows:

```
For each transaction $t_j \in$ partition of the uncertain Big data do
    for each item $x \in t_j$ do
        emit $\langle x, P(x, t_j)\rangle$.
```

This results in a list of different $\langle x, P(x, t_j)\rangle$ pairs.

Afterwards, these $\langle x, P(x, t_j)\rangle$ pairs are shuffled and sorted. Each processor then executes the *reduce function* on the shuffled and sorted pairs to apply constraint checking on every item x and obtain the expected support only for valid x (i.e., $\{x\}$ that satisfies C_{SAM}). In other words, the reduce function can be specified as follows:

```
For each $x \in \{\langle x$, list of $P(x, t_j)\rangle\}$ do
    if $\{x\}$ satisfying $C_{\text{SAM}}$ then
        set $expSup(\{x\}) = 0$;
        for each $P(x, t_j) \in$ list of $P(x, t_j)$ do
            $expSup(\{x\}) = expSup(\{x\}) + P(x, t_j)$.
        if $expSup(\{x\}) \geq minsup$ then emit $\langle \{x\}, expSup(\{x\})\rangle$.
```

To recap, when using a high-level description, this first set of "map" and "reduce" functions can be defined as follows:

$$\begin{aligned} \text{map}_1\colon & \ \langle \text{ID of transaction}\, t_j, \text{content of } t_j\rangle \mapsto \text{list of } \langle x, P(x, t_j)\rangle, \\ \text{reduce}_1\colon & \ \langle x, \text{list of } P(x, t_j)\rangle \mapsto \langle \text{valid frequent}\{x\}, expSup(\{x\})\rangle. \end{aligned}$$

Here, the master node first reads and divides uncertain Big data in partitions. The worker node corresponding to each partition then outputs $\langle x, P(x, t_j)\rangle$ pairs for each domain item x. The reduce function then sums all existential probability values of x for each x to compute its expected support in the probabilistic database of uncertain Big data.

Example 1. Consider a probabilistic database of uncertain data with auxiliary information as shown in Table 1. Let (i) $C_1 \equiv max(X.Temperature) \leq 20\,°\text{C}$, which means that domain items a, b, c, d (but not e) satisfy C_{SAM}, and (ii) *minsup*=1.0 Then, from transaction t_1, the map function outputs $\langle a, 0.7\rangle, \langle b, 1.0\rangle$, $\langle c, 0.8\rangle$ and $\langle e, 0.9\rangle$. Similarly, from transaction t_2, the map function outputs

Table 1. A sample set of uncertain Big data (with auxiliary information)

TID	Content
t_1	$\{a{:}0.7,\ b{:}1.0,\ c{:}0.8,\ e{:}0.9\}$
t_2	$\{a{:}0.9,\ b{:}1.0,\ c{:}0.6,\ e{:}0.7\}$
t_3	$\{a{:}0.8,\ c{:}0.2,\ d{:}0.8\}$

Item	Temp.	Item	Temp.
a	5°C	d	20°C
b	10°C	e	25°C
c	15°C		

$\langle a, 0.9\rangle$, $\langle b, 1.0\rangle, \langle c, 0.6\rangle$ and $\langle e, 0.7\rangle$; from transaction t_3, the map function outputs $\langle a, 0.8\rangle, \langle c, 0.2\rangle$ and $\langle d, 0.8\rangle$. These pairs are then shuffled and sorted. The reduce function reads $\langle a, [0.7, 0.9, 0.8]\rangle$, $\langle b, [1.0, 1.0]\rangle$, $\langle c, [0.8, 0.6, 0.2]\rangle$, $\langle d, [0.8]\rangle$ & $\langle e, [0.9, 0.7]\rangle$, and outputs $\langle a, 2.4\rangle, \langle b, 2.0\rangle$ &$\langle c, 1.6\rangle$ (i.e., valid frequent items and their corresponding expected support). Note that the reduce function does not sum the existential probabilities values for item e, let alone compute its expected support, because $\{e\}$ does not satisfy C_{SAM}. Although the reduce function sums the existential probabilities values for item d (as it satisfies C_{SAM}), it does not output its expected support because d is infrequent. □

3.2 Exploiting SAM Constraints in the Map Function

Alternatively, to handle the user-specified SAM constraint C_{SAM}, we can push C_{SAM} into the *map function* so that we only emit $\langle x, P(x, t_j)\rangle$ for each item $x \in t_j$ that satisfies C_{SAM}. See the corresponding map function:

```
For each transaction $t_j \in$ partition of the uncertain Big data do
    for each item $x \in t_j$ and $\{x\}$ satisfies $C_{\text{SAM}}$ do
        emit $\langle x, P(x, t_j)\rangle$.
```

This results in a list of different $\langle$valid $x, P(x, t_j)\rangle$ pairs.

Afterwards, these $\langle$valid $x, P(x, t_j)\rangle$ 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:

```
For each $x \in \{\langle$valid $x$, list of $P(x, t_j)\rangle\}$ do
    set $expSup(\{x\}) = 0$;
    for each $P(x, t_j) \in$ list of $P(x, t_j)$ do
        $expSup(\{x\}) = expSup(\{x\}) + P(x, t_j)$.
    if $expSup(\{x\}) \geq minsup$ then emit $\langle\{x\}, expSup(\{x\})\rangle$.
```

To recap, when using a high-level description, this alternative first set of "map" and "reduce" functions can be defined as follows:

$$\begin{aligned}
\text{map}_{1'}&: \langle \text{ID of } t_j, \text{content of } t_j\rangle \mapsto \text{list of } \langle \text{valid } x, P(x, t_j)\rangle,\\
\text{reduce}_{1'}&: \langle \text{valid } x, \text{list of } P(x, t_j)\rangle \mapsto \langle \text{valid frequent } \{x\}, expSup(\{x\})\rangle.
\end{aligned}$$

Example 2. Revisit Example 1. Again, let (i) domain items a, b, c, d (but not e) satisfy C_{SAM} and (ii) *minsup*=1.0. Then, from transaction t_1, the map function outputs $\langle a, 0.7\rangle$, $\langle b, 1.0\rangle$ and $\langle c, 0.8\rangle$. Similarly, from transaction t_2, the map function outputs $\langle a, 0.9\rangle$, $\langle b, 1.0\rangle$ and $\langle c, 0.6\rangle$; from transaction t_3, the map function outputs $\langle a, 0.8\rangle, \langle c, 0.2\rangle$ and $\langle d, 0.8\rangle$. Note that constraint checking was brought from the reduce function (as in Example 1) to the map function, which does not output the existential probability values for item e because $\{e\}$ does not satisfy C_{SAM}. All the output pairs are then shuffled and sorted. Afterwards, the reduce function reads $\langle a, [0.7, 0.9, 0.8]\rangle$, $\langle b, [1.0, 1.0]\rangle$, $\langle c, [0.8, 0.6, 0.2]\rangle$ & $\langle d, [0.8]\rangle$, and outputs $\langle a, 2.4\rangle, \langle b, 2.0\rangle$ & $\langle c, 1.6\rangle$ (i.e., valid frequent items and their corresponding expected support). Note that, although the reduce function sums the existential probabilities values for item d (as it satisfies C_{SAM}), it does not output its expected support because d is infrequent. □

As observed from the above two examples, exploiting SAM constraints in the reduce function requires fewer constraint checks because it only checks at most m domain items to see if they satisfy C_{SAM}. In contrast, exploiting SAM constraints in the map function checks all occurrences of items in every transaction in the Big data set, which are normally $\gg m$. Hence, the former is time-efficient when the data set consisting of only a few domain items such as DNA or RNA sequences in bioinformatics. On the other hand, the latter requires less bookkeeping because it emits $\langle \text{valid } x, P(x, t_j)\rangle$ only for those items that satisfy C_{SAM}. Hence, it is space-efficient when high volumes of data come at a high velocity such as data streams.

3.3 Mining Valid Frequent Non-singleton Patterns

Once our BigSAM algorithm finds valid frequent singleton patterns (with their associated expected support), it rereads each transaction in the probabilistic database of uncertain Big data to form an $\{x\}$-projected database (i.e., a collection of transactions containing x) for each valid frequent item x in the list returned by the first reduce function. Due to the succinctness &anti-monotonicity, all valid patterns must comprise only valid singleton items. Hence, our BigSAM algorithm keeps only those valid singleton items in each $\{x\}$-projected database. The worker node corresponding to each projected database then (i) builds appropriate local trees (e.g., UF-trees or PUF-trees)—based on the projected database assigned to the node—to mine every valid frequent pattern X of higher cardinality k (i.e., k-itemsets for $k \geq 2$) and (ii) outputs $\langle X, \mathit{expSup}(X)\rangle$ (i.e., every valid frequent pattern X with its expected support). In other words, BigSAM executes the second set of "map" and "reduce" functions as follows:

$$\begin{aligned}
\text{map}_2\text{: } & \langle \text{ID of } t_j, \text{content of } t_j\rangle \\
& \mapsto \text{list of } \langle \text{valid frequent}\{x\}, \text{prefix of } t_j \text{that ends with } x\rangle, \\
\text{reduce}_2\text{: } & \langle \text{valid frequent}\{x\}, \{x\} - \text{projected database}\rangle \\
& \mapsto \text{list of } \langle \text{valid frequent pattern } X, \mathit{expSup}(X)\rangle.
\end{aligned}$$

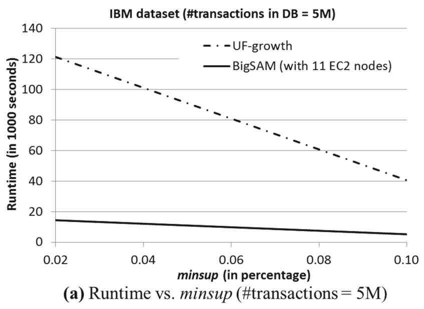
Example 3. Continue with Example 1 or 2. After reading t_1, BigSAM emits $\langle b, \{a{:}0.7, b{:}1.0\}\rangle$ & $\langle c, \{a{:}0.7, b{:}1.0, c{:}0.8\}\rangle$. BigSAM also emits $\langle b, \{a{:}0.9, b{:}1.0\}\rangle$ & $\langle c, \{a{:}0.9, b{:}1.0, c{:}0.6\}\rangle$ after reading t_2, and emits $\langle c, \{a{:}0.8, c{:}0.2\}\rangle$ after reading t_3. Note that the map function of BigSAM does not emit any pairs containing infrequent item d or invalid item e. After all the emitted pairs are shuffled and sorted, the reduce function of BigSAM then forms the $\{b\}$-projected database (comprising $\{a$:0.7, b:1.0$\}$ & $\{a$:0.9, b:1.0$\}$) and the $\{c\}$-projected database (comprising $\{a$:0.7, b:1.0, c:0.8$\}$, $\{a$:0.9, b:1.0, c:0.6$\}$ & $\{a$:0.8, c:0.2$\}$), from which valid frequent patterns $\{a, b\}$:1.8, $\{a, c\}$:1.26, $\{a, b, c\}$:1.1 and $\{b, c\}$:1.4 are found. □

4 Experimental Results

To evaluate our proposed BigSAM algorithm in mining uncertain Big data for frequent patterns that satisfy user-specified SAM constraints, we used different datasets—which include real-life datasets (e.g., accidents, connect4, and mushroom) from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/) and the FIMI repository (http://fimi.ua.ac.be/). We also used IBM synthetic data, which were generated using the IBM Quest Dataset Generator [1]. The generated data ranges from 2M to 5M transactions with an average transaction length of 10 items from a domain of 1K items. As these datasets originally contained only precise data, we assigned to each item in every transaction an existential probability value in the range (0,1]. All experiments were run on either (i) a single machine with an Intel Core i7 4-core processor (1.73 GHz) and 8 GB of main memory running a 64-bit Windows 7 operating system, or (ii) the Amazon EC2 cluster—specifically, 11 `m2.xlarge` computing nodes (http://aws.amazon.com/ec2/). All versions of the algorithm were implemented in the Java programming language. The stock version of Apache Hadoop 0.20.0 was used.

First, we used (i) a database consisting of items *all with existential probability value of 1* (indicating that all items are definitely present in the database and (ii) a user-specified SAM constraint. With this setting, we compared BigSAM (which finds valid frequent patterns from *uncertain* data) with CAP [21] (which also finds valid frequent patterns but from *precise* data). Experimental results showed that, (i) in terms of *accuracy*, BigSAM returned the *same* collection of valid frequent patterns as those returned by CAP. However, (ii) in terms of *functionality*, CAP is confined to finding valid frequent patterns from datasets when existential probability values of all items is 1, whereas our BigSAM algorithm is capable of finding valid frequent patterns from datasets containing items with *various existential probability values* ranging from 0 to 1.

Next, we used (i) a probabilistic database of uncertain data (containing items with various existential probability values ranging from 0 to 1) and (ii) a SAM constraint with 100 % selectivity (so that every item is selected). With this setting, we compared UF-growth [15] (which mines *unconstrained* frequent patterns from uncertain data) with BigSAM. Experimental results showed that, (i) in terms of *accuracy*, both algorithms returned the *same* collection of frequent patterns. (ii) In terms of *runtimes*, as shown in Fig. 1(a), both algorithms took shorter to run when *minsup* increased because fewer patterns had expected support $\geq$ *minsup*. (iii) Between the two algorithms, BigSAM (which was run in the MapReduce environment with 11 nodes) took much shorter runtimes than UF-growth [15] (which is a sequential algorithm). This led to a significant speedup,



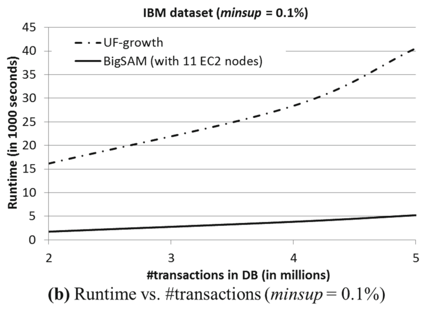


Fig. 1. UF-growth [15] vs. our BigSAM algorithm

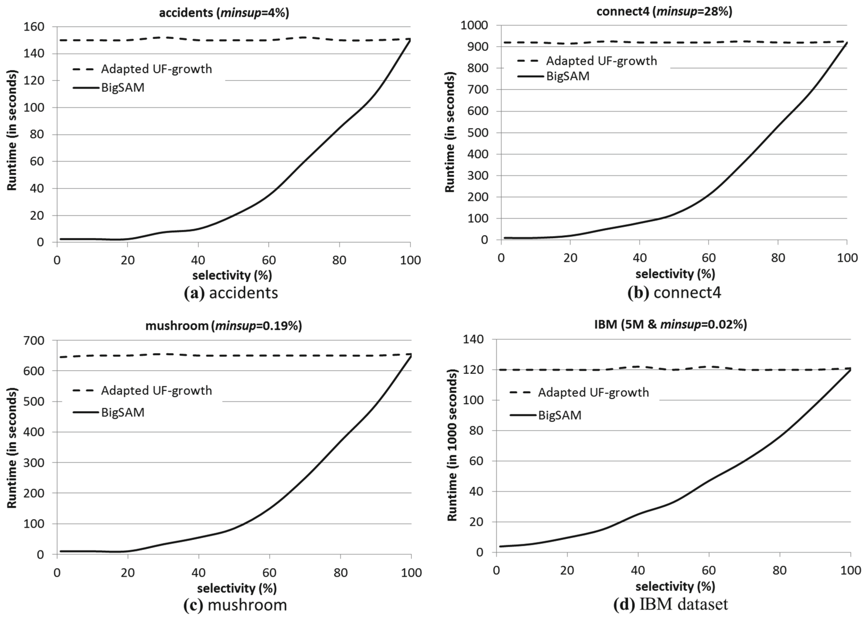


Fig. 2. Constraint handling by our BigSAM algorithm

especially for mining Big data. (iv) As observed from Fig. 1(b), when the number of transactions increased, the gap between the runtimes of the two algorithms increased, and thus the speedup became more significant. (v) In terms of *functionality*, UF-growth [15] was not designed to handle constraints with selectivity other than 100 %, whereas our BigSAM algorithm is capable of handling constraints of *any selectivity*.

To adapt UF-growth for handling user-specified SAM constraints with selectivity other than 100 %, it first ignores the constraints and mines all frequent patterns, and then applies constraint checking as a post-processing step to check if each mined pattern is valid (i.e., satisfying the SAM constraints) and prune those uninteresting/invalid patterns (i.e., patterns that violate the SAM constraints). By doing so, the computation of this adapted algorithm is independent of the selectivity of the SAM constraints. Hence, it would be time- and space-consuming for handling uncertain Big data—especially when only a few frequent patterns are valid (i.e., low percentage selectivity). In contrast, Fig. 2 shows that (i) runtimes of BigSAM decreased when the selectivity was lower (i.e., when fewer frequent patterns were valid) on different datasets. Moreover, (ii) the runtime was proportional to the computation, which was proportional to the percentage selectivity. This illustrates the benefits of pushing constraint checking in our BigSAM algorithm.

As ongoing work, we are conducting more experiments to evaluate other aspects (e.g., the effect on the number of machines or cluster nodes, the communication costs) and with other algorithms (e.g., PUF-growth [16]).

5 Conclusions

In this paper, we proposed the BigSAM algorithm that (i) allows users to express their interest in terms of succinct anti-monotone (SAM) constraints and (ii) uses the MapReduce programming model to mine uncertain Big data for frequent patterns that satisfy user-specified constraints. As a result, our algorithm returns all and only those patterns that are interesting to the users. Experimental results show the effectiveness of our algorithm in mining these interesting patterns from probabilistic databases of uncertain Big data with MapReduce. As ongoing work, we are extending our algorithm for handling non-SAM constraints.

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

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB 1994, pp. 487–499 (1994)
2. Calders, T., Garboni, C., Goethals, B.: Efficient pattern mining of uncertain data with sampling. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010, Part I. LNCS, vol. 6118, pp. 480–487. Springer, Heidelberg (2010)
3. Chen, Y.-C., Ko, Y.-L., Peng, W.-C., Lee, W.-C.: Mining appliance usage patterns in smart home environment. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013, Part I. LNCS (LNAI), vol. 7818, pp. 99–110. Springer, Heidelberg (2013)
4. Chui, C.-K., Kao, B., Hung, E.: Mining frequent itemsets from uncertain data. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 47–58. Springer, Heidelberg (2007)
5. Condie, T., Mineiro, P., Polyzotis, N., Weimer, M.: Machine learning for Big data. In: ACM SIGMOD 2013, pp. 939–942 (2013)
6. Cordeiro, R.L.F., Traina, C., Traina, A.J.M., López, J., Kang, U., Faloutsos, C.: Clustering very large multi-dimensional datasets with MapReduce. In: ACM KDD 2011, pp. 690–698 (2011)

7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. CACM **51**(1), 107–113 (2008)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1–12 (2000)
9. Koufakou, A., Secretan, J., Reeder, J., Cardona, K., Georgiopoulos, M.: Fast parallel outlier detection for categorical datasets using MapReduce. In: IEEE IJCNN 2008, pp. 3298–3304 (2008)
10. Kumar, A., Niu, F., Ré, C.: Hazy: making it easier to build and maintain big-data analytics. CACM **56**(3), 40–49 (2013)
11. Leung, C.K.-S.: Frequent itemset mining with constraints. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1179–1183. Springer, New York (2009)
12. Leung, C.K.S.: Big data mining and analytics. In: Wang, J. (ed.) Encyclopedia of Business Analytics and Optimization, pp. 328–337. IGI Global, Hershey (2014)
13. Leung, C.K.-S.: Mining uncertain data. WIREs Data Min. Knowl. Discov. **1**(4), 316–329 (2011)
14. Leung, C.K.-S., Hayduk, Y.: Mining frequent patterns from uncertain data with mapreduce for big data analytics. In: Meng, W., Feng, L., Bressan, S., Winiwarter, W., Song, W. (eds.) DASFAA 2013, Part I. LNCS, vol. 7825, pp. 440–455. Springer, Heidelberg (2013)
15. Leung, C.K.-S., Mateo, M.A.F., Brajczuk, D.A.: A tree-based approach for frequent pattern mining from uncertain data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 653–661. Springer, Heidelberg (2008)
16. Leung, C.K.-S., Tanbeer, S.K.: PUF-Tree: a compact tree structure for frequent pattern mining of uncertain data. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013, Part I. LNCS (LNAI), vol. 7818, pp. 13–25. Springer, Heidelberg (2013)
17. Lin, M.-Y., Lee, P.-Y., Hsueh, S.-C.: Apriori-based frequent itemset mining algorithms on MapReduce. In: ICUIMC 2012, Article 76 (2012)
18. Luo, W., Chan, K.C.C.: Discovering patterns in drug-protein interactions based on their fingerprints. BMC Bioinform. **13**(S–9), S4 (2012)
19. MacKinnon, R.K., Leung, C.K.-S., Tanbeer, S.K.: A scalable data analytics algorithm for mining frequent patterns from uncertain data. In: Peng, W.-C., Wang, H., Bailey, J., Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P. (eds.) PAKDD 2014 Workshops, LNCS (LNAI), vol. 8643, pp. 404-416. Springer, Heidelberg (2014)
20. Madden, S.: From databases to big data. IEEE Internet Comput. **16**(3), 4–6 (2012)
21. Ng, R.T., Ng, Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: ACM SIGMOD 1998, pp. 13–24 (1998)
22. Riondato, M., DeBrabant, J.A., Fonseca, R., Upfal, E.: PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In: ACM CIKM 2012, pp. 85–94 (2012)
23. Tang, L.-Y., Hsiu, P.-C., Huang, J.-L., Chen, M.-S.: iLauncher: an intelligent launcher for mobile apps based on individual usage patterns. In: ACM SAC 2013, pp. 505–512 (2013)
24. Yang, S., Wang, B., Zhao, H., Wu, B.: Efficient dense structure mining using MapReduce. In: IEEE ICDM Workshops 2009, pp. 332–337 (2009)
25. Zaki, M.J.: Parallel and distributed association mining: a survey. IEEE Concurrency **7**(4), 14–25 (1999)