Balancing Tree Size and Accuracy in Fast Mining of Uncertain Frequent Patterns

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

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

Abstract. To mine frequent patterns from uncertain data, many existing algorithms (e.g., UF-growth) directly calculate the expected support of a pattern. Consequently, they require a significant amount of storage space to capture all existential probability values among the items in the data. To reduce the amount of required storage space, some existing algorithms (e.g., PUF-growth) combine nodes with the same item by storing an upper bound on expected support. Consequently, they lead to many false positives in the intermediate mining step. There is trade-off between storage space and accuracy. In this paper, we introduce a new algorithm called MUF-growth for achieving a tighter upper bound on expected support than PUF-growth while balancing the storage space requirement. We evaluate the trade-off between storing more information to further tighten the bound and its effect on the performance of the algorithm. Our experimental results reveal a diminishing return on performance as the bound is increasingly tightened, allowing us to make a recommendation on the most effective use of extra storage towards increasing the efficiency of the algorithm.

1 Introduction and Related Works

Frequent pattern mining from precise data has become popular since the introduction of the *Apriori algorithm* [1]. Users definitely know whether an item is present in, or absent from, a transaction in databases of precise data. Common pattern mining techniques applied to precise data can be extended to the mining of interaction patterns [6], sequential patterns [7], social patterns [10], and popular patterns [17]. However, there are also situations in which users are uncertain about the presence or absence of items [3,4,9,20]. For example, a physician may highly suspect (but cannot guarantee) that a coughing patient suffers from the Middle East respiratory syndrome (MERS). The uncertainty of such suspicion can be expressed in terms of existential probability (e.g., a 70 % likelihood of suffering from the MERS). 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 handle uncertain data, the tree-based *UF-growth algorithm* [15] was proposed. In order to compute the *exact* expected support of each pattern, paths in the corresponding UF-tree are shared only if tree nodes on the paths have the

S. Madria and T. Hara (Eds.): DaWaK 2015, LNCS 9263, pp. 57–69, 2015.
DOI: 10.1007/978-3-319-22729-0_5

same item and same existential probability. Hence, due to this more restrictive path sharing requirement, the UF-tree may be quite large when compared to the FP-tree [8] for precise data. This issue has been addressed [13,14,16]. For instance, Calders et al. [5] mined uncertain data with sampling. Expected support of a pattern X is then estimated based on the average actual support of X in several random samples (or instantiations). We [18] previously proposed the *PUF-growth algorithm* to utilize a concept of *item caps* (which provide upper bounds to expected support) together with aggressive path sharing (in which paths are shared if nodes have the same item in common regardless of existential probability) to yield a more compact tree. Here, the expected support of X is estimated based on relevant existential probability values.

In this paper, we study the following questions: Can we further tighten the upper bound on expected support in PUF-growth? Can the resulting tree still be as compact as the FP-tree in terms of number of nodes? At what point does continuing to tighten the upper bound no longer provide an acceptable decrease in runtime? Our *key contributions* of this paper are as follows:

1. a **m**etal value **u**ncertain **f**requent pattern **tree (MUF-tree)**, which can be as compact as the original FP-tree;
2. a mining algorithm — called **MUF-growth** — that is guaranteed to find *all and only those* frequent patterns with *no* false negatives (frequent patterns mistakenly categorized as infrequent) and *no* false positives (infrequent patterns mistakenly categorized as frequent) from uncertain data at the end of the mining process; and
3. an empirical analysis of the upper bound in MUF-growth.

The remainder of this paper is organized as follows. The next section presents background. Then, we present our MUF-tree structure and MUF-growth algorithm in Sects. 3 and 4, respectively. Experimental results are shown in Sect. 5, and conclusions are given in Sect. 6.

2 Background

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 $= \{t_1, t_2, \ldots, t_n\}$ is the set of n transactions, where each transaction $t_j \subseteq$ `Item`. The projected database of X is the set of all transactions containing X. Each item x_i in a transaction $t_j = \{y_1, y_2, \ldots, y_h\}$ in an uncertain database is associated with an *existential probability value* $P(y_i, t_j)$, which represents the likelihood of the presence of y_i in t_j. Note that $0 < P(y_i, t_j) \leq 1$. The *existential probability* $P(X, t_j)$ of a pattern X in t_j is then the product of the corresponding existential probability values of items within X when these items are independent [11,12]: $P(X, t_j) = \prod_{x \in X} P(x, t_j)$. The *expected support* $expSup(X)$ of X in the database is the sum of $P(X, t_j)$ over all n transactions in the database: $expSup(X) = \sum_{j=1}^{n} P(X, t_j)$. A pattern X

is *frequent* in an uncertain database if $expSup(X) \geq$ 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 collection of frequent patterns having expected support $\geq$ *minsup*.

3 Our MUF-tree Structure

Recall that PUF-growth utilizes an upper bound to expected support to first find all potentially frequent patterns, which include (i) true positives (i.e., patterns with expected support $\geq$ *minsup*) and (ii) false positives (i.e., patterns with expected support $<$ *minsup* but with upper bound $\geq$ *minsup*). Then, PUF-growth verifies each of the patterns and returns only those truly frequent ones. To further tighten the upper bound to expected support, we propose (i) the **m**etal value **u**ncertain **f**requent pattern **tree (MUF-tree)** structure in this section and (ii) the corresponding **MUF-growth** algorithm in the next section.

The key idea behind the MUF-tree is to keep track of the i-th highest probability values in the prefix of t_j and use them every time a frequent extension ($k > 2$) is added to the suffix item during the mining process. Hence, each node in an MUF-tree keeps (i) an item y_r, (ii) an *item cap* $IC(y_r, t_j)$, and (iii) a list of *"metal" values*, which are the i-th highest existential probabilities in the prefix of y_r in t_j. Figure 1(c) shows the contents of an MUF-tree for the database in Table 1, in which each node maintains (i) an item, (ii) its item cap, and (iii) its first three metal values (i.e., "silver value" $M_2(y_r, t_j)$, "bronze value" $M_3(y_r, t_j)$

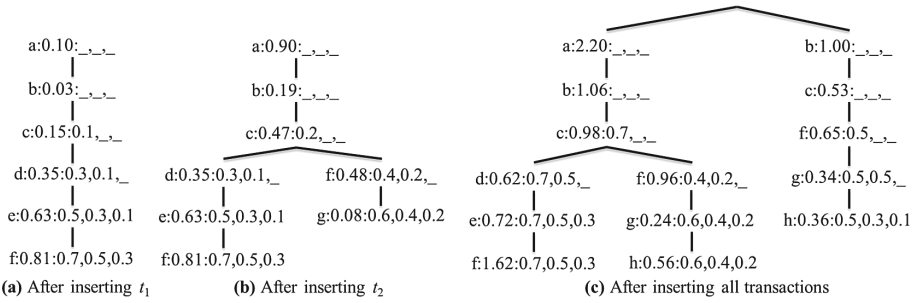

Fig. 1. Our MUF-tree for the database in Table 1 with *minsup* = 0.5 and 3 metal values

Table 1. A transactional database of uncertain data

Transactions	Transactions
$t_1 = \{a{:}0.1,\ b{:}0.3,\ c{:}0.5,\ d{:}0.7,\ e{:}0.9,\ f{:}0.9\}$	$t_4 = \{b{:}0.3,\ c{:}0.6,\ f{:}0.5,\ g{:}0.1,\ h{:}0.6\}$
$t_2 = \{a{:}0.8,\ b{:}0.2,\ c{:}0.4,\ f{:}0.6,\ g{:}0.1\}$	$t_5 = \{a{:}0.4,\ b{:}0.6,\ c{:}0.1,\ f{:}0.8,\ g{:}0.2,\ h{:}0.7\}$
$t_3 = \{b{:}0.7,\ c{:}0.5,\ f{:}0.5,\ g{:}0.4\}$	$t_6 = \{a{:}0.9,\ b{:}0.7,\ c{:}0.5,\ d{:}0.3,\ e{:}0.1,\ f{:}0.9\}$

and "copper value" $M_4(y_r, t_j)$). With this new data structure, a new *tighter* upper bound can be given by the product of $IC(y_r, t_j)$ and the various "metal" values. The *compounded* (*prefixed*) *item cap* — denoted as $CIC(X, t_j)$ — of any k-itemset $X = \{x_1, x_2, \ldots, x_k\}$ in $t_j = \{y_1, y_2, \ldots, y_r, \ldots, y_h\}$ (where $x_k = y_r$) can then be defined as in Definition 1.

Definition 1. *Let* $t_j = \{y_1, y_2, \ldots, y_{r-1}, y_r, \ldots, y_h\}$ *where* $h = |t_j|$. *Also, let* $M_i(x_r, t_j)$ *be the i-th metal value, which is the i-th highest existential probability value among all* $(r-1)$ *items in the proper prefix* $\{y_1, y_2, \ldots, y_{r-1}\} \subset t_j$. *If* $X = \{x_1, x_2, \ldots, x_k\}$ *is a k-itemset in* t_j *such that* $x_k = y_r$, *then*

$$CIC(X, t_j) = \begin{cases} IC(X, t_j) & if\ k = 2, \\ IC(X, t_j) \times \prod_{i=2}^{k-1} M_i(y_r, t_j) & if\ k \geq 3, \end{cases} \quad (1)$$

where $IC(X, t_j) = P(y_r, t_j) \times M_1(y_r, t_j)$. □

It is interesting to observe that the item cap $IC(X, t_j)$ provided by PUF-growth is a special case of the compounded item cap $CIC(X, t_j)$ provided by MUF-growth when $k = 2$ (i.e., where no metal values are stored). The generalization we present here benefits us by obtaining an increasingly tighter bound when multiplying successive metal values for larger itemsets $k \geq 3$. For example, when $k = 4$, $CIC(X, t_j)$ involves $IC(X, t_j)$, as well M_2 ("silver value") and M_3 ("bronze value"). Moreover, in theory, the maximum number of metal values we can use is ultimately limited above by the length of the largest transaction in the database. In practice, we will usually use a smaller number. See Sect. 5.

Since the expected support of X is the sum of all existential probabilities of X over all the transactions containing X, the *cap* of expected support of X can then be defined as follows.

Definition 2. *The cap of expected support — denoted as* $expSup^{Cap}(X)$ *— of a pattern* $X = \{x_1, \ldots, x_k\}$ *(where* $k > 1$*) is defined as the sum (over all n transactions in a database) of all the compounded item caps of* x_k *in all the transactions that contain X:* $expSup^{Cap}(X) = \sum_{j=1}^{n} \{CIC(X, t_j) \mid X \subseteq t_j\}$. □

Based on Definition 2, $expSup^{Cap}(X)$ for any k-itemset $X = \{x_1, \ldots, x_k\}$ can be considered as an *upper bound* to the expected support of X, i.e., $expSup(X) \leq expSup^{Cap}(X)$. So, if $expSup^{Cap}(X) < minsup$, then X cannot be frequent. Conversely, if X is a frequent pattern, then $expSup^{Cap}(X) \geq minsup$. We take advantage of this property to safely mine all frequent patterns using $expSup^{Cap}(X)$ in place of $expSup(X)$.

Lemma 1. *The cap of expected support of a pattern X satisfies the partial downward closure property: All non-empty subsets Y of a frequent pattern X (such that X and Y share the same suffix item* y_r*) are also frequent.* □

An MUF-tree can be constructed as follows. With the first scan of the uncertain database, we find all distinct *frequent* items. Then, the MUF-tree is constructed with the second database scan in a fashion similar to that of the FP-tree.

A key difference is that, when inserting a transaction item, we compute its item cap and its metal values. Only frequent items are inserted into the MUF-tree, and they are inserted according to some canonical order. If a node containing that item already exists in a tree path, in addition to updating its item cap, we also update its metal values by taking the *maximum* of each computed metal value and the corresponding existing value. Otherwise, we create a new node with this item cap and metal values. Similar to the FP-tree, our MUF-tree maintains horizontal node traversal pointers whose links are adjusted whenever a new node is created. For a better understanding of the MUF-tree construction, see Example 1.

Example 1. Consider the uncertain database in Table 1. Let the user-specified support threshold *minsup* be set to 0.5 and the number of stored metal values be equal to three. For simplicity, we arrange items in alphabetical order. After the first database scan, the expected supports of all items (after removing infrequent single items) are a:2.2, b:2.3, c:2.6, d:1.0, e:1.0, f:4.2, g:0.8 and h:1.3.

With the second database scan, we insert only the frequent items of each transaction (with their respective item cap and metal values). For instance, as shown in Fig. 1(a), when inserting transaction $t_1 = \{a$:0.1, b:0.3, c:0.5, d:0.7, e:0.9, f:0.9$\}$, we store:

1. a:0.10:_:_:_ because $IC(a, t_1) = P(a, t_1) = 0.10$;
2. b:0.03:_:_:_ because $IC(b, t_1) = P(b, t_1) \times M_1(b, t_1) = 0.3 \times 0.1 = 0.03$;
3. c:0.15:0.1:_:_ because $IC(c, t_1) = P(c, t_1) \times M_1(c, t_1) = 0.5 \times 0.3 = 0.15$ and $M_2(c, t_1) = 0.1$;
4. d:0.35:0.3:0.1:_ because $IC(d, t_1) = P(d, t_1) \times M_1(d, t_1) = 0.7 \times 0.5 = 0.35$, $M_2(d, t_1) = 0.3$ and $M_3(b, t_1) = 0.1$;
5. e:0.63:0.5:0.3:0.1 because $IC(e, t_1) = P(e, t_1) \times M_1(e, t_1) = 0.9 \times 0.7 = 0.63$, $M_2(e, t_1) = 0.5$, $M_3(e, t_1) = 0.3$ and $M_4(e, t_1) = 0.1$; as well as
6. f:0.81:0.7:0.5:0.3 because $IC(f, t_1) = P(f, t_1) \times M_1(f, t_1) = 0.9 \times 0.9 = 0.81$, $M_2(f, t_1) = 0.7$, $M_3(f, t_1) = 0.5$ and $M_4(f, t_1) = 0.3$.

As t_2 shares a common prefix $\langle a, b, c\rangle$ with an existing path in the MUF-tree created when t_1 was inserted, (i) the item cap values of those items in the common prefix are *added* to their corresponding nodes, and (ii) the metal values of those items are checked against the metal values for their corresponding nodes, with only the *maximum* saved for each node. In this case, $IC(a, t_2) = 0.8$ is added to the previous value of 0.1 to get 0.9. Similarly, $IC(b, t_2) = 0.16$ is added to the previous value of 0.03 to get 0.19. For c, $IC(c, t_2) = 0.32$ is added to the previous value of 0.15 to get 0.47. As $M_2(c, t_2) = 0.2$ is higher than the previous M_2 value of 0.1, the new value of 0.2 is stored. As f and g in t_2 not sharing any common prefix with t_1, new nodes are created as shown in Fig. 1(b). Figure 1(c) shows the status of the MUF-tree after inserting all of the remaining transactions. □

4 Our MUF-growth Algorithm

Here, we propose a pattern-growth mining algorithm called **MUF-growth**, which mines frequent patterns from our MUF-tree structure. Recall that the

construction of an MUF-tree is similar to that of a FP-tree, except that metal values are additionally stored. Thus, the basic operation in MUF-growth for mining frequent patterns is to construct a projected database for each potential frequent pattern and recursively mine its potentially frequent extensions.

Based on Lemma 1, we apply the MUF-growth algorithm to our MUF-tree for generating only those k-itemsets (where $k > 1$) with caps of expected support $\geq$ *minsup*. Similar to other algorithms (e.g., UFP-growth [2]), the mining process of our MUF-growth may also lead to some false positives in the intermediate step (e.g., at the end of the second database scan), but all these false positives will be filtered out with a quick third scan of the database. Hence, our MUF-growth is guaranteed to return to the user *all* and *only those* truly frequent patterns with *neither* false positives *nor* false negatives at the end of the mining process.

Example 2. The MUF-growth algorithm mines extensions of every frequet item. Consider the MUF-tree in Fig. 1(c). The $\{f\}$-conditional tree is created by accumulating all the prefix paths from each f node up to the root. In the very first recursion, each node in the prefix of each f node takes on the current value of $expSup^{Cap}(\{f\})$ since the cap of expected support already contains the frequency and existential probability information of f together with the highest existential probability value in the prefix of f. In addition, each node in the prefix of f in the global tree inherits the metal values of f when they appear in the $\{f\}$-conditional tree. Thus, the $\{f\}$-conditional tree, pictured in Fig. 2(a) consists of two branches, one for each prefix path of the f-nodes up to the root that is not shared. The left branch contains $\langle a, b, c, d, e\rangle$ with (i) d and e having an $expSup^{Cap}$ value of 1.62 and (ii) a, b and c have an $expSup^{Cap}$ value of $1.62 + 0.96 = 2.58$ due to the shared prefix between the two f-nodes. All the nodes in the left branch contain the metal values $\langle 0.7, 0.5, 0.3\rangle$ as they are (i) the default values for d and e by default and (ii) higher than its corresponding value in $\{0.4, 0.2, \_\}$ for a, b and c. The right branch contains $\langle b, c\rangle$, each having the $expSup^{Cap}$ value of 0.65 and the M_1 value of 0.5.

For the $\{f, c\}$-conditional tree in Fig. 2(b), the compounding effect of the metal values is observed because the $expSup^{Cap}$ value for each node in the prefix of c is further multiplied by the first metal value in each list for c. The left branch in the $\{f, c\}$- conditional tree contains $\langle a, b\rangle$ with an $expSup^{Cap}$ value of

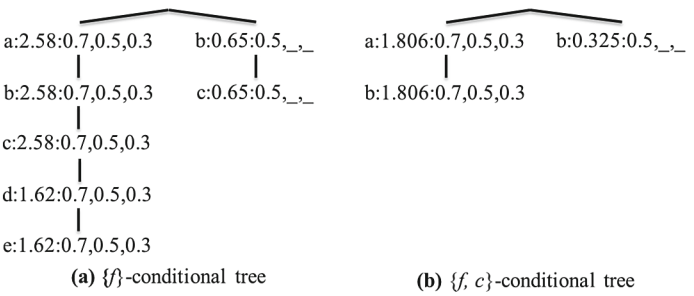

Fig. 2. Conditional trees in the MUF-growth algorithm

$1.806 = 2.58 \times 0.7$ and the metal values $\langle 0.7, 0.5, 0.3 \rangle$. The right branch contains just b with an $expSup^{Cap}$ value of $0.325 = 0.65 \times 0.5$ and the metal value 0.5 (which is just repeated as no further values were previously stored). At this point, if $minsup = 2.2$, then using just a single metal value M_2 has saved us having to further generate any more candidates which are extensions of $\{f, c\}$.

The $\{f, c, b\}$-conditional tree consists of a single node a:0.903:0.7,0.5,0.3. If we take advantage of the second metal value in b's list, each node in the prefix of b will take on the $expSup^{Cap}$ value of $0.903 = 1.806 \times 0.5$. If instead we only decided to store a single metal value in this implementation of MUF-growth, then we would reuse the first metal value in b's list. Consequently, each node in the prefix of b would take on the $expSup^{Cap}$ value of $1.2642 = 1.806 \times 0.7$. In this case, using extra storage space to accommodate more metal values in each node allows us to bring the $expSup^{Cap}$ value closer to the expected support with fewer recursive calls to generate candidate extensions. Every time an extra metal value is used in this manner, the chance goes up that the sub-tree generation stops earlier and fewer false positives are generated. □

Since each metal value used during the candidate generation process is a maximum among all values in the prefix of an item, MUF-growth finds a complete set of patterns from an MUF-tree without any false negatives. In addition, with each additional unique metal value that is stored in the MUF-tree nodes, MUF-growth does so while generating fewer false positives than existing algorithms. In much larger databases, the runtime savings caused by this effect are significant.

5 Evaluation Results

This section presents our results on evaluating the following aspects of our MUF-tree structure and its corresponding MUF-growth algorithm in mining frequent patterns from uncertain data.

5.1 Analytical Evaluation

Tree Compactness. For any uncertain databases, *our MUF-tree has the same number of nodes as in the existing PUF-tree* [18]. Thus, the node count of the MUF-tree (i) can be equal to that of the FP-tree [8] (when the MUF-tree is constructed using the frequency-descending order of items) and (ii) is bounded above by total number of frequent items summed over every transaction in the database.

Tree Completeness. The complete set of mining results (with no false positives and no false negatives) can be generated because the *MUF-tree contains the set of all of the frequent items from every transaction in the database along with the compounded item cap (CIC) in each node. Mining based on this compounded item cap ensures that no frequent k-itemset ($k > 1$) will be missed.*

Tightness of CIC. *Recall that the PUF-tree utilizes an item cap (IC), which is based on the existential probability value $P(y_r, t_j)$ of y_r and the single highest existential probability value $M_1(y_r, t_j)$ in its prefix. In contrast, our MUF-tree*

utilizes a CIC, which is based on various metal values $M_i(y_r, t_j)$ *in addition to* $P(y_r, t_j)$ *and* $M_1(y_r, t_j)$*.Note that the CIC used in our MUF-tree provides a tighter upper bound than the IC used in the PUF-tree* because candidates are generated during the mining process with increasing cardinality of X in the CIC (cf. the IC has no such compounding effect).

5.2 Empirical Evaluation

We also compared the performance of our MUF-growth algorithm (with varying numbers of metal values stored per node) with the existing PUF-growth algorithm. Recall that PUF-growth [18] was shown to outperform UF-growth [15], UFP-growth [2] and UH-Mine [2]. So, we compare our MUF-growth solely with PUF-growth in order to clearly evaluate the effect of the tightened upper bound. Similar to other papers [2] in the sub-community of uncertain frequent pattern mining, we also used both real life and synthetic databases for our tests. The synthetic databases, which are generally sparse, are generated within a domain of 1000 items by the data generator developed at IBM Almaden Research Center [1]. We also considered several real life databases such as mushroom, retail and kosarak. We assigned a (randomly generated) existential probability value from the range (0 %,100 %] to each item in every transaction in the databases. The name of each database indicates some of its characteristics. For example, the database u10k5L10100_2 contains 10 K transactions with average transaction length of 5, each item in a transaction is associated with an existential probability value that lies within a range of [10 %, 100 %] and the probability values are spaced out with minimum increment of 5 % (i.e., every 10 % increment is separated into 2 different values). Due to space constraints, we present here the results on some of the above databases. All programs were written in C++ and ran in a Linux environment on an Intel Core i5-661 CPU with 3.33 GHz and 8 GB of RAM. Unless otherwise specified, runtime includes CPU and I/Os for tree construction, mining, and false-positive removal. While the number of false positives generated at the end of the second database scan may vary, all algorithms (ours and others) produce the same set of truly frequent patters at the end of the mining process. The results shown in this section are based on the average of multiple runs for each case. In all experiments, *minsup* was expressed in terms of the absolute support value, and all trees were constructed using the descending order of expected support.

Number of False Positives. Although PUF-trees and MUF-trees are compact (in fact, the number of nodes in the global tree can be equal to the FP-tree for both of them), their corresponding algorithms generate some false positives. Hence, their overall performances depend on the number of false positives generated. In this experiment, we measured the number of false positives generated by both algorithms for fixed values of *minsup* with different databases. We present some results in Fig. 3 using one *minsup* value over several probability distributions for the synthetic database u10k5L10100 and the real-life database retail10100. In general, *MUF-growth was observed to remarkably reduce the number of false positives when compared with PUF-growth.* The primary reason for

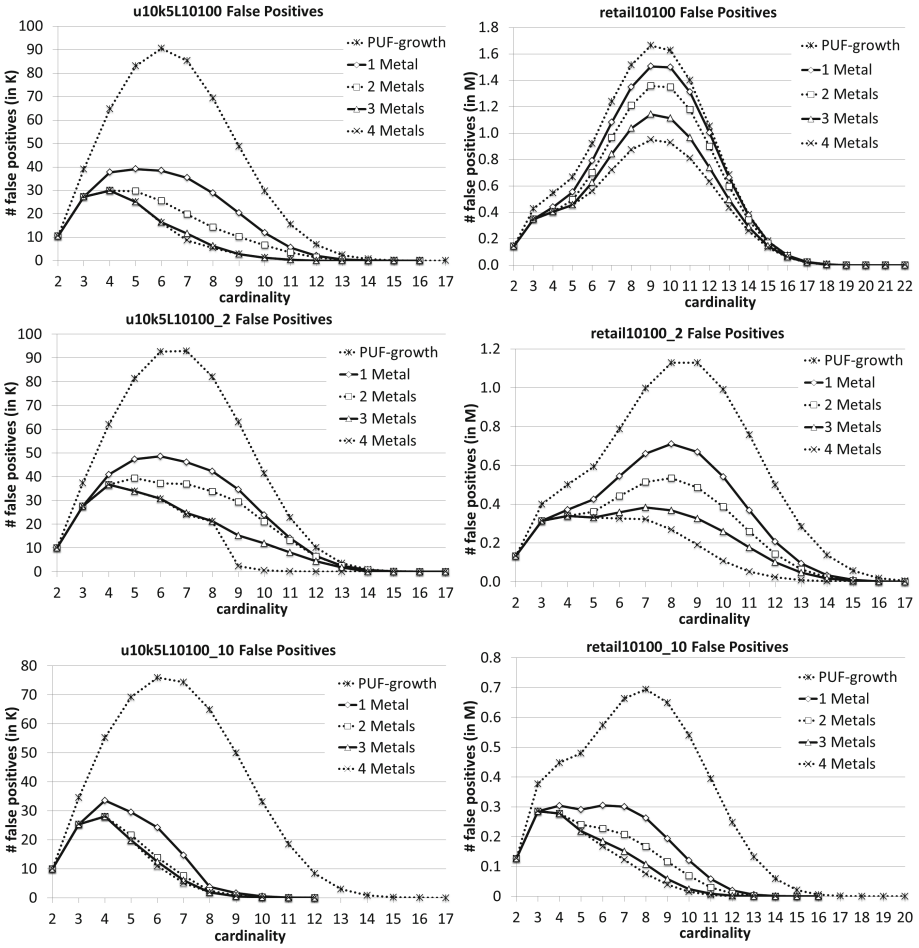

Fig. 3. Number of false positives

this improvement is that the upper bound in this algorithm is much tighter than that in PUF-growth for higher cardinality itemsets ($k > 2$), therefore less total candidates are generated and subsequently less false positives.

On the IBM synthetic database u10k5L10100, MUF-growth generated anywhere from 47 % to as low as 23 % of the false positives generated by the corresponding PUF-growth algorithm as the number of metal values was increased. On the other hand, with the UC Irvine real-life database retail10100, this effect is not quite as pronounced. When the number of distinct probability values is increased, as in the retail10100_10 database, we again see similar numbers as in the synthetic databases. The wider number of distinct probability values leads to a higher chance that additional metal values are significantly lower than the previous values. In synthetic data, the decrease in false positives is more pronounced even with fewer distinct values as the distribution of items in

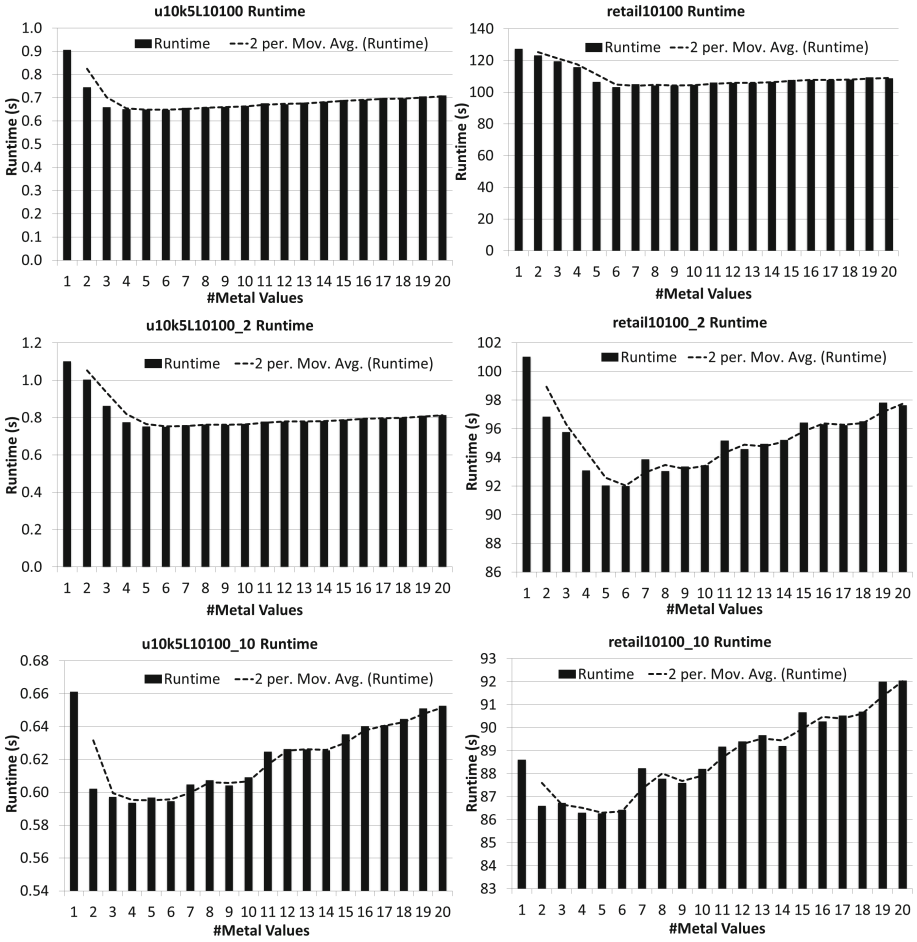

Fig. 4. Runtime

transactions becomes completely independent, whereas items in the retail database tend to appear or be absent in predetermined groups.

When changing the range of observed probability values, as in retail_5060 (not shown for brevity), we found that *MUF-growth generated fewer than 1.5 % of the total false positives of the corresponding PUF-growth algorithm.* Adding subsequent metal values beyond the first in retail_5060 did not make a significant difference in the number of false positives generated. The reason is that with only two closely spaced existential probability values to choose from, each additional metal value is either the exact same or very close to the previous one, giving little to no benefit over just reusing the previous value.

Runtime. Recall that PUF-growth was shown to outperform UH-Mine [18] and subsequently UFP-growth [2,19]. Hence, we compared our MUF-growth

algorithm with different metal values to PUF-growth. *The addition of just a single metal value in MUF-growth makes a remarkable improvement in the runtime when compared to PUF-growth.* The primary reason is that, even though PUF-growth finds the exact set of frequent patterns when mining an extension of X, it may suffer from the high computation cost of generating unnecessarily large numbers of infrequent candidates and their extensions. *The use of the metal values in MUF-growth ensure that those high cardinality candidates are never generated due to their expected support caps being much closer to the expected support.*

When comparing our MUF-growth algorithm to it with different metal values the runtimes are much closer together. We show, in Fig. 4, the runtime of MUF-growth on the same databases we used earlier in this section. With these databases *we notice a diminishing return on runtime after five or six metal values.* For instance, on the retail10100 database, MUF-growth showed a decrease in runtime from 127 seconds to 103 seconds as the number of metal values increased from one to six, followed by subsequent increases in runtime as additional metal values were used. With the u10k5L10100 databases, similar numbers of metals values provided a decrease in run time. On the other hand, with the retail_5060 database we notice a diminishing return on runtime after a smaller number of metal values. For all databases, at the point where the runtime starts increasing again, the upper bound given by the increased number of metal values is so close as to be indistinguishable from the bound obtained with fewer metal values, thus the extra computation is wasted.

Scalability. To test the scalability of MUF-growth, we mined frequent patterns from increasing sizes of input databases. The experimental results indicate that our algorithm (i) is scalable with respect to the number of transactions and (ii) can mine large volumes of uncertain data within a reasonable amount of time.

The experimental results show that our algorithm effectively mines frequent patterns from uncertain data irrespective of distribution of existential probability values (whether they are distributed into a narrow or wide range of values) and whether the data is dense or sparse.

6 Conclusions

In this paper, we proposed (i) the **MUF-tree structure** for capturing important information from uncertain data and (ii) the **MUF-growth algorithm** for mining frequent patterns from the MUF-tree structure. The algorithm obtains upper bounds on the expected supports of frequent patterns by accumulating item caps in its tree structure. These item caps are *compounded* with various metal values (computed based on the i-th highest existential probabilities of any item in the prefix) during the mining of potentially frequent patterns. Hence, they further tighten the upper bound on expected supports of frequent patterns when compared to the existing PUF-growth algorithm. Our proposed MUF-growth algorithm has been shown to generate significantly fewer false positives than

PUF-growth (e.g., up to 1 % of the total value). In all cases, MUF-growth significantly outperformed PUF-growth. Using extra metal values in MUF-growth to further tighten the upper bound was shown to be most effective on sparse data, providing further decreases in both false positives and runtime. Our algorithm is guaranteed to find *all* frequent patterns (with *no* false negatives). Experimental results show the effectiveness of our MUF-growth algorithm in fast mining of uncertain frequent patterns when balancing tree size and accuracy.

Acknowledgement. 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: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB 1994, pp. 487–499. Morgan Kaufmann, San Francisco (1994)
2. Aggarwal, C.C., Li, Y., Wang, J., Wang, J.: Frequent pattern mining with uncertain data. In: Elder, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) ACM KDD 2009, pp. 29–37. ACM, New York (2009)
3. Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., Zuefle, A.: Probabilistic frequent itemset mining in uncertain databases. In: Elder, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) ACM KDD 2009, pp. 119–127. ACM, New York (2009)
4. Calders, T., Garboni, C., Goethals, B.: Approximation of frequentness probability of itemsets in uncertain data. In: Webb, G.I., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (eds.) IEEE ICDM 2010, pp. 749–754. IEEE, Los Alamitos (2010)
5. 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 (LNAI), vol. 6118, pp. 480–487. Springer, Heidelberg (2010)
6. Fariha, A., Ahmed, C.F., Leung, C.K.-S., Abdullah, S.M., Cao, L.: Mining frequent patterns from human interactions in meetings using directed acyclic graphs. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013, Part I. LNCS (LNAI), vol. 7818, pp. 38–49. Springer, Heidelberg (2013)
7. Fournier-Viger, P., Gomariz, A., Šebek, M., Hlosta, M.: VGEN: fast vertical mining of sequential generator patterns. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2014. LNCS, vol. 8646, pp. 476–488. Springer, Heidelberg (2014)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Chen, W., Naughton, J.F., Bernstein, P.A. (eds.) ACM SIGMOD 2000, pp. 1–12. ACM, New York (2000)
9. Jiang, F., Leung, C.K.-S.: Stream mining of frequent patterns from delayed batches of uncertain data. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2013. LNCS, vol. 8057, pp. 209–221. Springer, Heidelberg (2013)
10. Jiang, F., Leung, C.K.-S., Liu, D., Peddle, A.M.: Discovery of really popular friends from social networks. In: IEEE BDCloud 2014, pp. 342–349. IEEE, Los Alamitos (2014)
11. Leung, C.K.-S.: Uncertain frequent pattern mining. In: Aggarwal, C.C., Han, J. (eds.) Frequent Pattern Mining, pp. 417–453. Springer, Switzerland (2014)

12. Leung, C.K.-S., Jiang, F.: A data science solution for mining interesting patterns from uncertain big data. In: IEEE BDCloud 2014, pp. 235–242. IEEE, Los Alamitos (2014)
13. Leung, C.K.-S., MacKinnon, R.K.: BLIMP: a compact tree structure for uncertain frequent pattern mining. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2014. LNCS, vol. 8646, pp. 115–123. Springer, Heidelberg (2014)
14. Leung, C.K.-S., MacKinnon, R.K., Tanbeer, S.K.: Fast algorithms for frequent itemset mining from uncertain data. In: Kumar, R., Toivonen, H., Pei, J., Huang, J.Z., Wu, X. (eds.) IEEE ICDM 2014, pp. 893–898. IEEE, Los Alamitos (2014)
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.: Fast tree-based mining of frequent itemsets from uncertain data. In: Lee, S., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012, Part I. LNCS, vol. 7238, pp. 272–287. Springer, Heidelberg (2012)
17. Leung, C.K.-S., Tanbeer, S.K.: Mining popular patterns from transactional databases. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 291–302. Springer, Heidelberg (2012)
18. 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)
19. Tong, Y., Chen, L., Cheng, Y., Yu, P.S.: Mining frequent itemsets over uncertain databases. PVLDB **5**(11), 1650–1661 (2012)
20. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: Wang, J.T.-L. (ed.) ACM SIGMOD 2008, pp. 819–832. ACM, New York (2008)