BLIMP: A Compact Tree Structure for Uncertain Frequent Pattern Mining

Carson Kai-Sang Leung⋆ and Richard Kyle MacKinnon

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

Abstract. Tree structures (e.g., UF-trees, UFP-trees) corresponding to many existing uncertain frequent pattern mining algorithms can be large. Other tree structures for handling uncertain data may achieve compactness at the expense of loose upper bounds on expected supports. To solve this problem, we propose a compact tree structure that captures uncertain data with tighter upper bounds than the aforementioned tree structures. The corresponding algorithm mines frequent patterns from this compact tree structure. Experimental results show the compactness of our tree structure and the tightness of upper bounds to expected supports provided by our uncertain frequent pattern mining algorithm.

1 Introduction and Related Works

Over the past few years, many frequent pattern mining algorithms have been proposed [7, 9, 13, 14], which include those mining uncertain data [3–5, 11, 16]. For instance, the *UF-growth algorithm* [10] is one of the uncertain frequent pattern mining algorithms. In order to compute the expected support of each pattern, paths in the corresponding UF-tree are shared only if tree nodes on the paths have the same item and same existential probability. Consequently, the UF-tree may be quite large when compared to the FP-tree [6] (for mining frequent patterns from precise data). In an attempt to make the tree compact, the *UFP-growth algorithm* [2] groups *similar* nodes (with the same item x and similar existential probability values) into a cluster. However, depending on the clustering parameter, the corresponding UFP-tree may be as large as the UF-tree (i.e., no reduction in tree size). Moreover, because UFP-growth does not store every existential probability value for an item in a cluster, it returns not only the frequent patterns but also some infrequent patterns (i.e., false positives). The *PUF-growth algorithm* [12] addresses these deficiencies by utilizing the idea of upper bounds to expected support with much more aggressive path sharing (in which paths are shared if nodes have the same item in common regardless of existential probability), to yield a final tree structure that can be as compact as the FP-tree is for precise data.

In this paper, we study the following questions: can we further tighten the upper bounds on expected support (e.g., than the PUF-tree)? Can the resulting

⋆ Corresponding author.

tree still be as compact as the FP-tree? How would frequent patterns be mined from such a tree? Would such a mining algorithm be faster than PUF-growth? Our *key contributions* of this paper are as follows:

1. a **b**ranch-**l**evel **i**te**m** **p**refixed-cap **tree (BLIMP-tree)**, which can be as compact as the original FP-tree and PUF-tree; and
2. a mining algorithm (namely, **BLIMP-growth**), which finds all frequent patterns from uncertain data.

The remainder of this paper is organized as follows. The next section presents background material. We then propose our BLIMP-tree structure and BLIMP-growth algorithm in Sections 3 and 4, respectively. Experimental results are shown in Section 5, and conclusions are given in Section 6.

2 Background

Let (i) `Item` be a set of m domain items and (ii) $X = \{x_1, x_2, \ldots, x_k\} \subseteq \texttt{Item}$ be a k-itemset (i.e., a pattern consisting of k items), where $1 \leq k \leq m$. Then, a transactional database $= \{t_1, t_2, \ldots, t_n\}$ is a set of n transactions. The projected database of X is the set of all transactions containing X. Each item x_i in a transaction $t_j = \{x_1, x_2, \ldots, x_h\} \subseteq \texttt{Item}$ in an uncertain database is associated with an **existential probability value $\boldsymbol{P(x_i, t_j)}$**, which represents the likelihood of the presence of x_i in t_j [8]. Note that $0 < P(x_i, t_j) \leq 1$. The **expected support $\boldsymbol{expSup(X)}$** of X in the database is the sum (over all n transactions) of the product of the corresponding existential probability values of items within X when these items are independent [8]: $expSup(X) = \sum_{j=1}^{n} \left(\prod_{x \in X} P(x, t_j)\right)$. Hence, given (i) a database of uncertain data and (ii) a user-specified minimum support threshold *minsup*, the research problem of *frequent pattern mining from uncertain data* is to discover from the database a complete set of *frequent* patterns (i.e., to discover every pattern X having $expSup(X) \geq minsup$).

To mine frequent patterns from uncertain data, the *PUF-growth algorithm* [12] scans the uncertain data to build a PUF-tree for capturing the contents of transactions in the uncertain data. Specifically, each node in a PUF-tree captures (i) an item x and (ii) its *prefixed item cap.*

Definition 1. The **prefixed item cap** [12] of an item x_r in a tree path $t_j = \langle x_1, \ldots, x_r, \ldots, x_h \rangle$ representing a transaction where $1 \leq r \leq h$—denoted as $\boldsymbol{I^{Cap}(x_r, t_j)}$—is defined as the product of (i) $P(x_r, t_j)$ and (ii) the highest existential probability value M_1 of items from x_1 to x_{r-1} in tree path t_j (i.e., in the *proper prefix* of x_r in t_j):

$$I^{Cap}(x_r, t_j) = \begin{cases} P(x_1, t_j) & \text{if } h = 1 \\ P(x_r, t_j) \times M_1 & \text{if } h > 1 \end{cases} \tag{1}$$

where $M_1 = \max_{1 \leq q \leq r-1} P(x_q, t_j)$. □

Table 1. A transactional database of uncertain data (*minsup*=1.1)

TID	Transactions
t_1	a:0.6, b:0.1, c:0.2, f:0.8, g:0.5
t_2	a:0.5, b:0.2, c:0.1, e:0.9, g:0.6
t_3	a:0.7, b:0.2, c:0.2, f:0.9
t_4	a:0.9, b:0.1, c:0.1, e:0.8, f:0.6
t_5	b:0.9, c:0.9, d:0.4

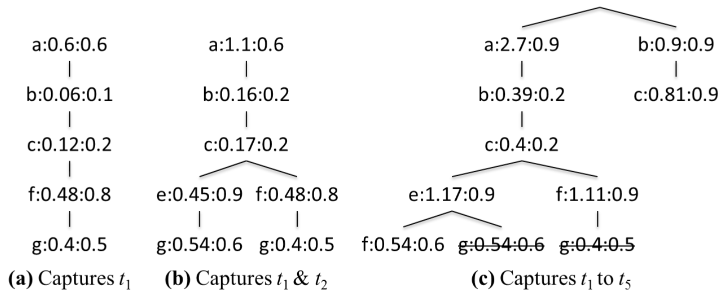


Fig. 1. BLIMP-trees for the database shown in Table 1 when *minsup*=1.1

It was proven [12] that the expected support of any k-itemset X (where $k > 2$) (which is the product of two or more probability values) must be $\leq I^{Cap}(x_r, t_j)$. However, such an upper bound may not be too tight when dealing with long patterns mined from long transactions of uncertain data. See Example 1. In many real-life situations, it is not unusual to have long patterns to be mined from long transactions of uncertain data.

Example 1. Consider a path $t_1 = \langle a{:}0.6, b{:}0.1, c{:}0.2, f{:}0.8, g{:}0.5\rangle$ in a PUF-tree capturing the contents of uncertain data shown in Table 1. If $X=\{a, b, c, f\}$, then $I^{Cap}(f, t_1) = P(f, t_1)\times M_1 = 0.8\times0.6 = 0.48$ because 0.6 is the highest existential probability value in the proper prefix $\langle a{:}0.6, b{:}0.1, c{:}0.2\rangle$ of f in t_1. Note that $I^{Cap}(f, t_1)$ also serves as an upper bound to the expected support of patterns $\{a, f\}, \{b, f\}, \{c, f\}, \{a, b, f\}, \{a, c, f\}, \{b, c, f\}$ and $\{a, b, c, f\}$. While this upper bound is tight for short patterns like $\{a, f\}$ having $P(\{a, f\}, t_1)$=0.48, it becomes loose for long patterns like $\{a, b, f\}$ having $P(\{a, b, f\}, t_1)$ =0.048 and $\{a, b, c, f\}$ having $P(\{a, b, c, f\}, t_1)$=0.0096. □

3 Our BLIMP-tree Structure

To tighten the upper bound for all k-itemsets ($k > 2$), we propose a **b**ranch-**l**evel **i**te**m** **p**refixed-cap **tree structure (BLIMP-tree)**. The key idea is to keep track of a value calculated solely from the maximum of all existential probabilities for the single item represented by that node. Every time a frequent extension is added to the suffix item to form a k-itemset (where $k > 2$), this "blimp" value will be used. Hence, each node in a BLIMP-tree contains: (i) an item x_r,

(ii) an item cap $I^{Cap}(x_r, t_j)$ and (iii) a "blimp" value, which is the maximum existential probability of x_r in t_j. Fig. 1(c) shows the contents of a BLIMP-tree for the database in Table 1.

With this information, BLIMP-trees give a *tightened upper bound* on the expected support of an itemset by the product of $I^{Cap}(x_r, t_j)$ and the "blimp" values in the prefix of x_r. This new *compounded item cap* of any k-itemset $X = \{x_1, x_2, \ldots, x_k\}$ in a tree path $t_j = \langle x_1, x_2, \ldots, x_h\rangle$ (denoted as $I(\widehat{X, t_j})$ where $x_k = x_r$) can be defined as follows.

Definition 2. Let $t_j = \langle x_1, x_2, \ldots, x_r, \ldots, x_h\rangle$ be a path in a BLIMP-tree, where $h = |t_j|$ and $r \in [1, h]$. Let M_{x_i} denote the highest existential probability of x_i in the prefix of x_r in t_j. If $X = \{x_1, x_2, \ldots, x_k\}$ is a k-itemset in t_j such that $x_k = x_r$, its **compounded item cap $\boldsymbol{I(\widehat{X, t_j})}$** is defined as follows:

$$I(\widehat{X, t_j}) = \begin{cases} I^{Cap}(x_r, t_j) & \text{if } k \leq 2 \\ I^{Cap}(x_r, t_j) \times \prod_{i=1}^{k-2} M_{x_i} & \text{if } k \geq 3 \end{cases} \tag{2}$$

where $I^{Cap}(x_r, t_j)$ is the prefixed item cap as defined in Definition 1. □

Example 2. Let us revisit Example 1. If X=$\{a, b, c, f\}$, then $I(\widehat{X, t_1}) = I^{Cap}(f, t_1) \times (\prod_{i=1}^{2} M_{x_i}) = (0.8 \times M_1) \times (M_{x_1} \times M_{x_2}) = (0.8 \times 0.6) \times (0.6 \times 0.1) = 0.0288$. Simiarly, if X'=$\{b, c, g\}$, then $I(\widehat{X', t_1}) = I^{Cap}(g, t_1) \times (\prod_{i=1}^{1} M_{x_i}) = (0.5 \times M_1) \times M_{x_1} = (0.5 \times 0.8) \times 0.1 = 0.04$. □

To construct a BLIMP-tree, we scan the transactional database of uncertain data to compute the expected support of every domain item. Any infrequent items are removed. Then, we scan the database a second time to insert each transaction into the BLIMP-tree. An item is inserted into the BLIMP-tree according to a predefined order. If a node containing that item already exists in the tree path, we (i) update its item cap by *summing* the current $I^{Cap}(x_r, t_j)$ with the existing item cap value and (ii) update its "blimp" value by taking the *maximum* of the current $P(x_r, t_j)$ with the existing "blimp" value. Otherwise, we create a new node with $I^{Cap}(x_r, t_j)$ and $P(x_r, t_j)$ (i.e. the initial "blimp" value). For a better understanding of BLIMP-tree construction, see Example 3.

Example 3. Consider the database in Table 1, and let *minsup*=1.1. For simplicity, items are arranged in the alphabetic order. After the first database scan, we remove infrequent domain item d. The remaining items a:2.7, b:1.4, c:1.4, e:1.7, f:2.3 & g:1.1 are frequent. With the second database scan, we insert only the frequent items of each transaction (with their respective item cap and "blimp" values). For instance, after reading transaction t_1=$\{a$:0.6, b:0.1, c:0.2, f:0.8, g:0.5$\}$, we insert $\langle a$:0.6:0.6, b:0.06(=0.1×0.6):0.1, c:0.12(=0.2×0.6):0.2, f:0.48(=0.8×0.6):0.8, g:0.4(=0.5×0.8):0.5$\rangle$ into the BLIMP-tree as shown in Fig. 1(a). As the tree path for t_2 shares a common prefix $\langle a, b, c\rangle$ with an existing path in the BLIMP-tree created when t_1 was inserted, (i) the item cap values of those items in the common prefix (i.e., a, b and c) are added to their corresponding nodes, (ii) the existential probability values of those items are checked against the "blimp" values for their corresponding nodes, with only the maximum saved for each node, and (iii) the remainder of the transaction (i.e., a new branch for items e

and g) is inserted as a child of the last node of the prefix (i.e., as a child of c). See Fig. 1(b). Fig. 1(c) shows the BLIMP-tree after inserting all transactions and pruning those items with infrequent extensions (i.e. item g because its total item cap is less than *minsup*). □

Observation 1. With this compact BLIMP-tree, we observed the following: (a) $expSup^{Cap}(X)$, which sums compunded item caps for tree paths containing X, serves as an upper bound on the expected support of X. (b) $expSup^{Cap}(X)$ does *not* generally satisfy the downward closure property because $expSup^{Cap}(Y)$ can be less than $expSup^{Cap}(X)$ for some proper subset Y of X. (c) For special cases where X' and its subset Y' share the same suffix item (e.g., $Y'=\{a,b,f\} \subset \{a,b,c,f\}=X'$ sharing the suffix item f), $expSup^{Cap}(Y')$ for BLIMP-trees satisfies the downward closure property. (d) The number of tree nodes in a BLIMP-tree can be equal to that of an FP-tree [6] (when the BLIMP-tree is constructed using the frequency-descending order of items). (e) The compounded item cap $I(\widehat{X, t_j})$ computed based on the existential probability value of x_k, the highest existential probability value in its prefix and the "blimp" values of its prefix items provides a *tighter* upper bound than that based on the non-compounded item cap $I^{Cap}(x_r, t_j)$ of PUF-trees because the former tightens the bound as candidates are generated during the mining process with increasing cardinality of X, whereas the latter has no such compounding effect. □

4 The BLIMP-growth Algorithm

To mine frequent patterns (from our BLIMP-tree structure), we propose a tree-based pattern-growth mining algorithm called **BLIMP-growth**. The basic operation in BLIMP-growth is to construct a projected database for each potential frequent pattern and recursively mine its potentially frequent extensions.

Once an item x is found to be potentially frequent, the existential probability of x must contribute to the expected support computation for every pattern constructed from the $\{x\}$-projected database (denoted as DB_x). Hence, the complete set of patterns with suffix x can be mined (ref. Observation 1(c)). Let (i) X be a k-itemset (where $k > 1$) with $expSup^{Cap}(X) \geq minsup$ in the database and (ii) Y be an itemset in DB_X. Then, $expSup^{Cap}(Y \cup X)$ in the original database $\geq minsup$ if and only if $expSup^{Cap}(Y)$ in all the transactions in $DB_X \geq minsup$. Like UFP-growth [2] and PUF-growth [12], this mining process may also lead to some false positives (i.e., those itemsets that appear to be frequent but are truly infrequent) in the resulting set of frequent patterns at the end of the second database scan.

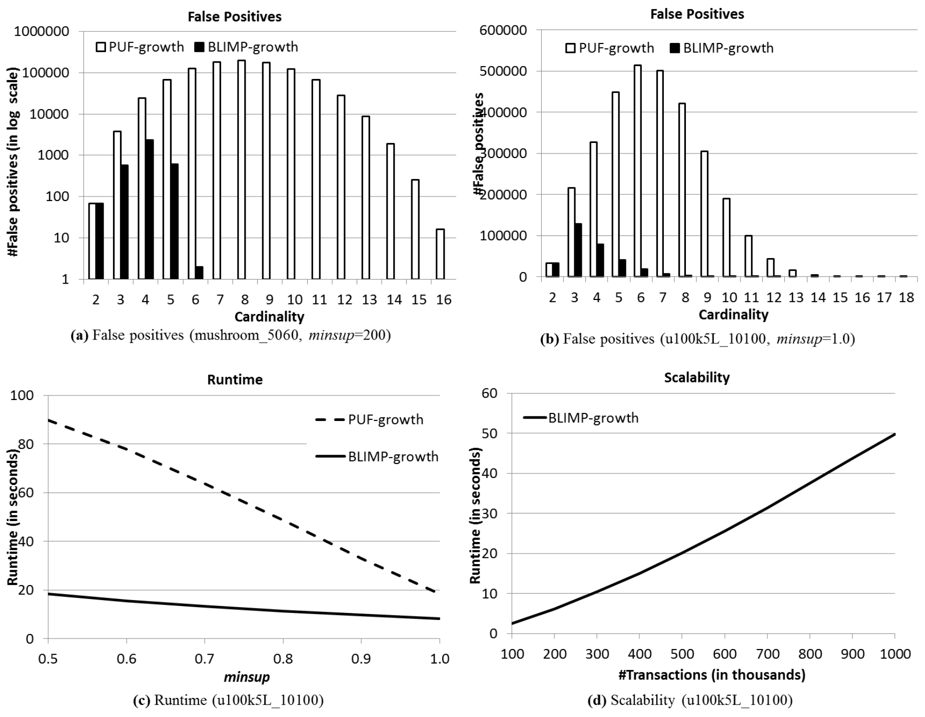
Fortunately, all these false positives will be filtered out with a third database scan. Hence, our BLIMP-growth is guaranteed to return to the user the *exact* collection of frequent patterns (i.e., *all* and *only those* frequent patterns with *neither* false positives *nor* false negatives).

5 Experimental Results

As it was shown [12] that PUF-growth outperformed many existing algorithms (e.g., UF-growth [10], UFP-growth [2] and UH-Mine [2]), we compared the performances of our BLIMP-growth algorithm with PUF-growth. We used both real life and synthetic datasets for our tests. The synthetic datasets, 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 datasets such as kosarack, mushroom and retail. We assigned a (randomly generated) existential probability value from the range (0,1] to each item in every transaction in the dataset. The name of each dataset indicates some characteristics of the dataset. For example, the dataset u100k5L_10100 contains 100K transactions with average transaction length of 5, and each item in a transaction is associated with an existential probability value that lies within a range of [10%, 100%].

All programs were written in C++ and ran in a Linux environment on an Intel Core i5-661 CPU with 3.33GHz and 7.5GB 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 ascending order of item value.

False Positives. Although PUF-trees and BLIMP-trees are compact (in fact, the number of nodes in the global tree can be equal to the FP-tree for both), 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 datasets. Figs. 2(a)–(b) shows the results when using one *minsup* value for each of the two datasets (i.e., mushroom_5060 and u100k5L_10100). BLIMP-growth was observed to greatly reduce the number of false positives when compared with PUF-growth. The primary reason of this improvement is that the upper bounds for the BLIMP-growth algorithm are much tighter than PUF-growth for higher cardinality itemsets ($k > 2$), hence less total candidates are generated and subsequently less false positives. If fact, when existential probability values were distributed over a narrow range with a higher *minsup* as shown in Fig. 2(a), BLIMP-growth generated fewer than 1% of the total false positives of PUF-growth. When the existential probability values were distributed over a wider range with a much lower *minsup*, as in Fig. 2(b), the total number of false positives in BLIMP-growth was still fewer than 10% of the total false positives of PUF-growth. As a result, BLIMP-growth had a runtime less than or equal to that of PUF-growth in every single experiment we ran.



(a) False positives (mushroom_5060, *minsup*=200)

(b) False positives (u100k5L_10100, *minsup*=1.0)

(c) Runtime (u100k5L_10100)

(d) Scalability (u100k5L_10100)

Fig. 2. Experimental results

Runtime. Recall that PUF-growth was shown [12, 15] to outperform UH-Mine and subsequently UFP-growth. Hence, we compared our BLIMP-growth algorithm with PUF-growth. Fig. 2(c) shows that, for low values of *minsup*, BLIMP-growth had shorter runtimes than PUF-growth for u100k5L_10100. 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 candidates due to only using two values in its item cap calculation: the existential probability of the suffix item and the single highest existential probability value in the prefix of x_r in t_j. This allows large amounts of high cardinality candidates to be generated with similar expected support cap values as low cardinality candidates with the same suffix item. The use of the "blimp" values in BLIMP-growth ensures that those high cardinality candidates are never generated due to their expected support caps being much closer to the actual expected support. Moreover, for lower values of *minsup*, the number of high cardinality candidates being generated increases. In this situation, the probability is higher that the "blimp" values in each node will actually be low, tightening the upper bound even further.

Scalability. To test the scalability of BLIMP-growth, we applied the algorithm to mine frequent patterns from datasets with increasing size. The experimental results presented in Fig. 2(d) indicates 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 algorithms effectively mine frequent patterns from uncertain data irrespective of distribution of existential probability values (whether most of them have low or high values and whether they are distributed into a narrow or wide range of values).

6 Conclusions

In this paper, we proposed a compact tree structure—called *BLIMP-tree*—for capturing important information from uncertain data. In addition, we presented the *BLIMP-growth algorithm* for mining frequent patterns from the BLIMP-tree. The algorithm obtains upper bounds on the expected supports of frequent patterns based on the *compounded item caps*. As these item caps are compounded with a *"blimp" value* (computed based on the maximum existential probability of a particular item), they further tighten the upper bound on expected supports of frequent patterns when compared to PUF-growth. BLIMP-growth has been shown to generate significantly fewer false positives than PUF-growth (e.g., 1% of the total value). Our algorithms are guaranteed to find *all* frequent patterns (with *no* false negatives). As ongoing work, we are conducting theoretical analyses on the tightness of the upper bound.

Acknowledgements. 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. Aggarwal, C.C., Li, Y., Wang, J., Wang, J.: Frequent pattern mining with uncertain data. In: ACM KDD 2009, pp. 29–37 (2009)
3. Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., Zuefle, A.: Probabilistic frequent itemset mining in uncertain databases. In: ACM KDD 2009, pp. 119–127 (2009)
4. Calders, T., Garboni, C., Goethals, B.: Approximation of frequentness probability of itemsets in uncertain data. In: IEEE ICDM 2010, pp. 749–754 (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. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1–12 (2000)
7. 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)

8. Leung, C.K.-S.: Mining uncertain data. WIREs Data Mining and Knowledge Discovery 1(4), 316–329 (2011)
9. Leung, C.K.-S., Hao, B.: Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE 2009, pp. 1663–1670 (2009)
10. 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)
11. Leung, C.K.-S., Tanbeer, S.K.: Fast tree-based mining of frequent itemsets from uncertain data. In: Lee, S.-G., 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)
12. 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)
13. Oguz, D., Ergenc, B.: Incremental itemset mining based on matrix Apriori algorithm. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 192–204. Springer, Heidelberg (2012)
14. Qu, J.-F., Liu, M.: A fast algorithm for frequent itemset mining using Patricia* structures. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 205–216. Springer, Heidelberg (2012)
15. Tong, Y., Chen, L., Cheng, Y., Yu, P.S.: Mining frequent itemsets over uncertain databases. PVLDB 5(11), 1650–1661 (2012)
16. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: ACM SIGMOD 2008, pp. 819–832 (2008)