A Tree-Based Approach for Frequent Pattern Mining from Uncertain Data

Carson Kai-Sang Leung*, Mark Anthony F. Mateo, and Dale A. Brajczuk

The University of Manitoba, Winnipeg, MB, Canada
{kleung, mfmateo, umbrajcz}@cs.umanitoba.ca

Abstract. Many frequent pattern mining algorithms find patterns from traditional transaction databases, in which the content of each transaction—namely, items—is definitely known and precise. However, there are many real-life situations in which the content of transactions is uncertain. To deal with these situations, we propose a tree-based mining algorithm to efficiently find frequent patterns from uncertain data, where each item in the transactions is associated with an existential probability. Experimental results show the efficiency of our proposed algorithm.

1 Introduction

Over the past decade, there have been numerous studies [1,2,3,6,7,8,9,11,12,13,14,15] on mining frequent patterns from *precise* data such as databases of market basket transactions, web logs, and click streams. In these databases of precise data, users definitely know whether an item (or an event) is present in, or is absent from, a transaction in the databases. However, there are situations in which users are uncertain about the presence or absence of some items or events [4,5,10]. For example, a physician may highly suspect (but cannot guarantee) that a patient suffers from flu. The uncertainty of such suspicion can be expressed in terms of *existential probability*. So, in this uncertain database of patient records, each transaction t_i represents a visit to the physician's office. Each item within t_i represents a potential disease, and is associated with an existential probability expressing the likelihood of a patient having that disease in t_i. For instance, in t_i, the patient has an 80% likelihood of having the flu, and a 60% likelihood of having a cold regardless of having the flu or not. With this notion, each item in a transaction t_i in traditional databases containing precise data can be viewed as an item with a 100% likelihood of being present in t_i.

Since there are many real-life situations in which data are uncertain, *efficient algorithms for mining uncertain data* are in demand. To mine frequent patterns from *uncertain* data, Chui et al. [4] proposed an algorithm called *U-Apriori*. Although they also introduced a trimming strategy to reduce the number of candidates that need to be counted, their algorithm is Apriori-based (i.e., relies on the candidate generate-and-test paradigm). Hence, some natural questions to ask are: Can we avoid generating candidates at all? Since tree-based algorithms for handling precise data [8,13] are usually faster than their Apriori-based counterparts [1,9], is this also the case when handling

* Corresponding author.

T. Washio et al. (Eds.): PAKDD 2008, LNAI 5012, pp. 653–661, 2008.

uncertain data? In response to these questions, we did a feasibility study [10] on using a tree for mining uncertain data. The study showed that the tree can be used for uncertain data mining. Hence, in the current paper, we propose an efficient tree-based algorithm for mining uncertain data. The **key contributions** of our work are (i) the proposal of an effective tree structure—called a *UF-tree*—for capturing the content of transactions consisting of uncertain data, (ii) the development of an efficient algorithm—called *UF-growth*—for mining frequent patterns from the proposed tree, and (iii) two improvements to the proposed UF-growth algorithm for mining frequent patterns from the UF-tree. Experimental results in Section 5 show the effectiveness of our proposed algorithm in mining frequent patterns from uncertain data.

This paper is organized as follows. The next section gives related work and background. In Section 3, we introduce our UF-growth algorithm for mining frequent patterns from uncertain data. Improvements to this UF-growth algorithm are described in Section 4. Section 5 shows experimental results. Finally, conclusions are presented in Section 6.

2 Related Work and Background

Both the Apriori algorithm [1] and the FP-growth algorithm [8] were designed to handle *precise* data—but not *uncertain* data. A key difference between precise and uncertain data is that each transaction of the latter contains items and their *existential probabilities*. The existential probability $P(x, t_i)$ of an item x in a transaction t_i indicates the likelihood of x being present in t_i. Using the *"possible world" interpretation* of uncertain data [4,5], there are two possible worlds for an item x and a transaction t_i: (i) W_1 where $x \in t_i$ and (ii) W_2 where $x \notin t_i$. Although it is uncertain which of these two worlds be the true world, the probability of W_1 be the true world is $P(x, t_i)$ and that of W_2 is $1 - P(x, t_i)$. To a further extent, there are many items in each of many transactions in a transaction database *TDB*. Hence, the *expected support* of a pattern (or a set of items) X in *TDB* can be computed by summing the support of X in possible world W_j (while taking in account the probability of W_j to be the true world) over all possible worlds:

$$expSup(X) = \sum_j \left[sup(X) \text{ in } W_j \times \prod_{i=1}^{|TDB|} \left(\prod_{x \in t_i \text{ in } W_j} P(x, t_i) \times \prod_{y \notin t_i \text{ in } W_j} (1 - P(y, t_i)) \right) \right] \quad (1)$$

$$= \sum_{i=1}^{|TDB|} \left(\prod_{x \in X} P(x, t_i) \right). \quad (2)$$

With this setting, a pattern X is considered *frequent* if its expected support equals or exceeds the user-specified support threshold *minsup*.

To handle uncertain data, Chui et al. [4] proposed the **U-Apriori algorithm**, which is a modification of the Apriori algorithm. Specifically, instead of incrementing the support counts of candidate patterns by their *actual* support, U-Apriori increments the support counts of candidate patterns by their *expected* support (using Equation (2)). As indicated by Chui et al., U-Apriori suffers from the following problems: (i) Inherited from the Apriori algorithm, U-Apriori does not scale well when handling large amounts

of data because it also follows a levelwise generate-and-test framework. (ii) If the existential probabilities of most items within a pattern X are small, increments for each transaction can be insignificantly small. Consequently, many candidates would not be recognized as infrequent until most (if not all) transactions were processed.

3 Our Proposed UF-Growth Algorithm

In this section, we propose a tree-based algorithm, called **UF-growth**, for mining uncertain data. The algorithm consists of two main operations: (i) the construction of UF-trees and (ii) the mining of frequent patterns from UF-trees.

3.1 Construction of the UF-Tree

As with many tree-based mining algorithms, a key challenge here is how to represent and store data—in this case, uncertain data—in a tree? For precise data, each item in a database transaction *TDB* is implicitly associated with a definite certainty of its presence in the transaction. In contrast, for uncertain data, each item is explicitly associated with an *existential probability* ranging from a positive value close to 0 (indicating that the item has an insignificantly low chance to be present in *TDB*) to a value of 1 (indicating that the item is definitely present). Moreover, the existential probability of the item can vary from one transaction to another. Different items may have the same existential probability.

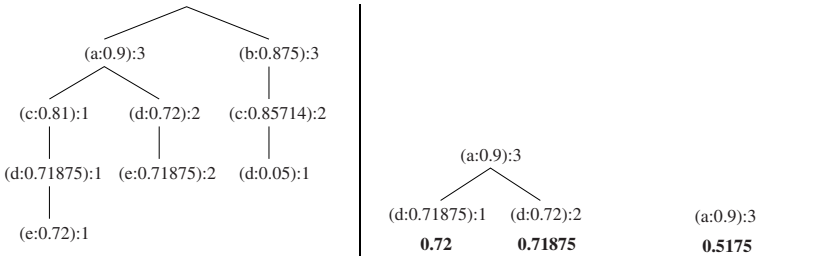
To effectively represent uncertain data, we propose a **UF-tree** which is a variant of the FP-tree. Each node in our UF-tree stores (i) an item, (ii) its expected support, and (iii) the number of occurrence of such expected support for such an item. Our proposed UF-growth algorithm constructs the UF-tree as follows. It scans the database once and accumulates the expected support of each item. Hence, it finds all frequent items (i.e., items having expected support $\geq$ *minsup*). It sorts these frequent items in descending order of accumulated expected support. The algorithm then scans the database the second time and inserts each transaction into the UF-tree in a similar fashion as in the construction of an FP-tree except for the following:

– The new transaction is merged with a child (or descendant) node of the root of the UF-tree (at the highest support level) only if the same item *and the same expected support* exist in both the transaction and the child (or descendant) nodes.

With such a tree construction process, our UF-tree possesses a nice property that *the occurrence count of a node is at least the sum of occurrence counts of all its children nodes*. See Example 1 for an illustration on constructing a UF-tree.

Example 1. Consider the following transaction database TDB consisting of uncertain data:

Transactions	Contents
t_1	$\{a$:0.9, d:0.72, e:$\frac{23}{32}$=0.71875, f:0.8$\}$
t_2	$\{a$:0.9, c:0.81, d:0.71875, e:0.72$\}$
t_3	$\{b$:$\frac{7}{8}$=0.875, c:$\frac{6}{7}\approx$0.85714$\}$
t_4	$\{a$:0.9, d:0.72, e:0.71875$\}$
t_5	$\{b$:0.875, c:0.85714, d:0.05$\}$
t_6	$\{b$:0.875, f:0.1$\}$



(a) The UF-tree for TDB

(b) The UF-tree for $\{e\}$-projected DB

(c) The UF-tree for $\{d, e\}$-projected DB

Fig. 1. The UF-trees

Here, each transaction contains items and their corresponding existential probability (e.g., the existential probability of item a in transaction t_1 is 0.9).

Let the user-specified support threshold *minsup* be set to 1. The UF-tree can be constructed as follows. First, our UF-growth algorithm scans TDB once and accumulates the expected support of each item. Hence, it finds all frequent items and sorts them in descending order of (accumulated) expected support. Items a, b, c, d and e are frequent (i.e., expected support of each of these items $\geq$ *minsup*), with their corresponding accumulated expected support of 2.7, 2.625, 2.52429, 2.20875 and 2.1575. Item f having accumulated expected support of $0.9 <$ *minsup* is removed because it is infrequent.

Then, UF-growth scans TDB the second time and inserts each transaction into the UF-tree. The algorithm first inserts the content of t_1 into the tree, and results in a tree branch $\langle(a$:0.9):1, (d:0.72):1, (e:0.71875):1$\rangle$. It then inserts the content of t_2 into the UF-tree. Since the expected support of a in t_2 is the same as the expected support of a in an existing branch (i.e., the branch for t_1), this node can be shared. So, UF-growth increments the occurrence count for the tree node (a:0.9) to 2, and adds the remainder of t_2 as a child of the node (a:0.9):2. As a result, we get the tree branch $\langle(a$:0.9):2, (c:0.81):1, (d:0.71875):1, (e:0.72):1$\rangle$. Afterwards, UF-growth inserts the content of t_3 as a new branch $\langle(b$:0.875):1, (c:0.85714):1$\rangle$ because the node (b:0.875):1 cannot be shared with the node (a:0.9):2. Remaining three transactions (t_4 to t_6) are then inserted into the UF-tree in a similar fashion. Consequently, at the end of the tree construction process, we get the UF-tree shown in Fig. 1(a) capturing the content of the above TDB of uncertain data. □

3.2 Mining of Frequent Patterns from the UF-Tree

Once the UF-tree is constructed, our UF-growth algorithm recursively mines frequent patterns from this tree in a similar fashion as in the FP-growth algorithm except for the following:

- Our UF-growth uses UF-trees (instead of FP-trees) for mining.
- When forming a UF-tree for the projected database for a pattern X, we need to keep track of the expected support (in addition to the occurrence) of X.

– When computing the expected support of an extension of a pattern X (say, $X \cup \{y\}$), we need to multiply the expected support of y in a tree path by the expected support of X.

See Example 2 for an illustration on how the UF-growth algorithm finds frequent patterns from the UF-tree.

Example 2. Once the UF-tree for Example 1 is constructed, our proposed UF-growth algorithm recursively mines frequent patterns from this tree with *minsup*=1 as follows. It starts with item e (with $expSup(\{e\})$ = 2.1575). UF-growth extracts from two tree paths—namely, (i) $\langle$(a:0.9), (c:0.81), (d:0.71875)$\rangle$ occurs once with (e:**0.72**) and (ii) $\langle$(a:0.9), (d:0.72)$\rangle$ occurs twice with (e:**0.71875**)—and forms the $\{e\}$-projected DB. Then, $expSup(\{a, e\})$ = (1 × **0.72** × 0.9) + (2 × **0.71875** × 0.9) = 1.94175, and $expSup(\{d, e\})$ = (1 × **0.72** × 0.71875) + (2 × **0.71875** × 0.72) = 1.5525. So, both patterns $\{a, e\}$ and $\{d, e\}$ are frequent. However, $\{c, e\}$ is infrequent because $expSup(\{c, e\})$ = 1 × **0.72** × 0.81 < *minsup*. Thus, c is removed from the $\{e\}$-projected DB. The UF-tree for this $\{e\}$-projected DB is shown in Fig. 1(b).

Then, the UF-growth algorithm extracts from the UF-tree for the $\{e\}$-projected DB to form the $\{d, e\}$-projected DB, which consists of $\{a\}$ (which represents the frequent pattern $\{a, d, e\}$) with $expSup(\{a, d, e\})$ = 3 × **0.5175** × 0.9 = 1.39725, where 0.5175 = 0.71875×0.72 represents $expSup(\{d, e\})$ in this projected DB. The UF-tree for this $\{d, e\}$-projected DB is shown in Fig. 1(c).

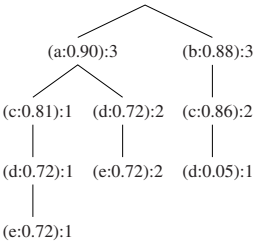
Next, UF-growth deals with items d, c and b (and finds all frequent supersets of $\{d\}$, $\{c\}$ and $\{b\}$) in a similar fashion. Consequently, by applying our proposed UF-growth algorithm to the UF-tree that captures the content of uncertain data in Example 1, we find frequent patterns $\{a\}, \{a, d\}, \{a, d, e\}, \{a, e\}, \{b\}, \{b, c\}, \{c\}, \{d\}, \{d, e\}$ and $\{e\}$. □

4 Improvements to Our Proposed UF-Growth Algorithm

The UF-tree above may appear to require a large amount of memory. Due to nature of uncertain data, the UF-tree is often larger than the FP-tree. This is because the FP-tree merges nodes sharing the same item whereas the UF-tree merges nodes sharing both the same item and the *same expected support*, where the expected support is in the domain of real numbers in the range of (0,1]—which can be infinitely many. Hence, the chance of sharing a path in the FP-tree is higher than that in the UF-tree. However, it is important to note that, even in the worst case, the number of nodes in a UF-tree is the same as the sum of the number of items in all transactions in the original database of uncertain data. Moreover, thanks to advances in modern technology, we are able to make the same realistic assumption as in many studies [3,7,15] that *we have enough main memory space* in the sense that the trees can fit into the memory.

A natural question to ask is: Can we reduce the memory consumption? In this section, we discuss how we improve our proposed UF-growth algorithm.

Improvement 1. To reduce the memory consumption and to increase the chance of path sharing, we discretize and round the expected support of each tree node up to k decimal places (e.g., 2 decimal places). By so doing, we reduce the potentially infinite number of possible expected support values—in the domain of real numbers in



(a) The improved UF-tree for TDB

(b) The improved UF-tree for $\{e\}$-projected DB

Fig. 2. The improved UF-trees (with Improvement 1)

the range of (0,1]—to a maximum of 10^k possible values (e.g., at most 100 possible expected support values ranging from 0.01 to 1.00 inclusive when $k = 2$). Thus, sizes of the UF-trees for the original TDB and subsequent projected databases are reduced. Fig. 2 shows some of these smaller UF-trees when Improvement 1 is applied.

Improvement 2. Inspired by the idea of the co-occurrence frequent-itemset tree [6], we modify and improve the mining procedure in our proposed UF-growth algorithm so that UF-trees are built *only* for the first two levels (i.e., for the original TDB and for each singleton pattern). In other words, the improved UF-growth does *not* need to build subsequent UF-trees for any non-singleton patterns (e.g., *not* need to build a UF-tree for the $\{d, e\}$-projected database). Specifically, the improved UF-growth systematically extracts relevant paths from the UF-tree built for each singleton, enumerates all subsets of each extracted tree path, summing the expected support of patterns extracted from these paths to find frequent patterns. See Example 3 for an illustration on how the improved UF-growth algorithm finds frequent patterns from the UF-tree.

Example 3. Similar to Example 2, the improved UF-growth builds a UF-tree for the original TDB (as illustrated in Example 1), finds frequent singleton patterns (e.g., $\{a\}, \{b\}, \{c\}, \{d\}$ and $\{e\}$), forms a projected DB and builds a UF-tree for each of these singletons starting with singleton $\{e\}$. From the $\{e\}$-projected DB, the improved UF-growth does *not* build any subsequent trees such as $\{d, e\}$-projected DB. Instead, the algorithm first extracts the tree path $\langle(a$:0.9):3, (d:0.71875):1$\rangle$ that occurs once with (e:**0.72**), enumerates all its subsets and obtains $\{a, e\}, \{a, d, e\}$ & $\{d, e\}$ (with their expected supports equal 0.648, 0.46575 & 0.5175 so far), and then decrements the occurrence counts of all nodes in this path. The algorithm then extracts the tree path $\langle(a$:0.9):2, (d:0.72):2$\rangle$ that occurs twice with (e:**0.71875**), enumerates all its subsets and obtains $\{a, e\}, \{a, d, e\}$ & $\{d, e\}$ (with their accumulative expected supports equal 1.94175, 1.39725 & 1.5525), and then decrements the occurrence counts of all nodes in this path. Afterwards, all the nodes have occurrence counts equal to 0. We find frequent patterns $\{a, e\}, \{a, d, e\}$ & $\{d, e\}$ and their expected supports, directly from the UF-tree representing the $\{e\}$-projected DB and *without* forming any subsequent UF-trees for non-singletons. Our improved UF-growth applies this technique to other UF-trees for singletons and finds other frequent patterns in a similar fashion. As a result, it finds the same set of frequent patterns as in Example 2 but requires less memory space. □

Note that Improvement 2 can be applied independently or in conjunction with Improvement 1 (i.e., rounding expected support values).

5 Experimental Results

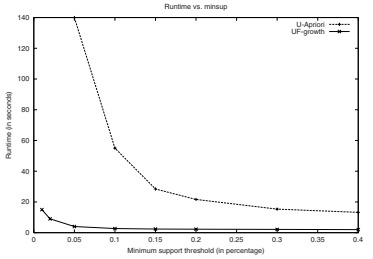
We conducted the following experiments using various databases including the IBM synthetic datasets [1], real-life databases from the UC Irvine Machine Learning Depository, as well as datasets from the Frequent Itemset Mining Implementation (FIMI) Dataset Repository. The experimental results were consistent. Hence, for lack of space, we only show below the experimental results on the IBM datasets, which contain 100k records with an average transaction length of 10 items and a domain of 1,000 items. We assigned an existential probability from the range (0,1] to each item in each transaction. All experiments were run in a time-sharing environment on a 1 GHz machine. The reported results are based on the average of multiple runs. Runtime includes CPU and I/Os; it includes the time for both tree construction and frequent pattern mining steps. In the experiments, we mainly evaluated the efficiency of the proposed algorithm.

First, we tested the effect of *minsup*. Theoretically, (i) the runtime decreases when *minsup* increases and (ii) our UF-growth algorithm (which does *not* rely on the candidate generate-and-test paradigm) requires much less runtime than the U-Apriori algorithm [4] (which relies on the candidate generation process). Experimental results (as shown in Fig. 3(a)) confirmed that, when *minsup* increased, fewer patterns had expected support $\geq$ *minsup*, and thus shorter runtimes were required. Moreover, our tree-based mining algorithm (UF-growth) outperformed its Apriori-based counterpart (U-Apriori).

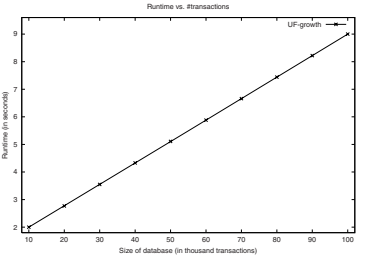
Second, we tested scalability of our proposed UF-growth algorithm. Theoretically, UF-growth should be scalable with respect to the number of transactions. Experimental results (as shown in Fig. 3(b)) confirmed that mining with our proposed algorithm had linear scalability.

Third, we tested the effect of the distribution of item existential probability. Theoretically, when items take on many different existential probability values, UF-trees (for the original TDB, projected databases for singletons as well as for non-singletons) become larger and times for both UF-tree construction and frequent pattern mining become longer. On the other hand, when items take on a few unique existential probability values, the runtime becomes shorter. This is confirmed by experimental results (as shown in Fig. 3(c)). Note that we can reduce the number of unique existential probability values by applying Improvement 1.

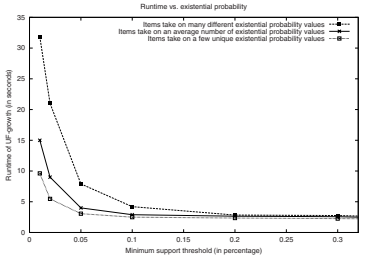
Fourth, we measured the number of nodes in UF-trees. Theoretically, our proposed UF-growth described in Section 3 builds UF-trees for the original TDB and projected databases for singletons as well as for non-singletons. The total number of nodes in the UF-tree representing the original TDB is no more than the total number of items in all transactions in TDB. The size of this tree, as well as other UF-trees, built for UF-growth with Improvement 1 is the same—and usually smaller than—that without improvement. Moreover, UF-growth with Improvement 2 builds *only* UF-trees representing the original TDB and projected databases for singletons; it does *not* build any UF-trees representing projected databases for non-singleton patterns. See Table 1 and Fig. 3(d). The graph shows the reduction in tree size when k=2 decimal places were



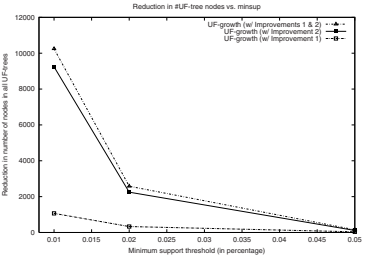
(a) Runtime vs. minsup



(b) Runtime vs. DB size



(c) Runtime vs. existential probability



(d) Reduction in tree size

Fig. 3. Experimental results on our proposed UF-growth algorithm

Table 1. Comparison on the sizes of UF-trees for variants of the UF-growth algorithm

UF-trees for...	UF-growth	w/ Improvement 1	w/ Improvement 2	w/ Improvements 1 & 2
TDB	$\#nodes_{TDB}$	$\leq \#nodes_{TDB}$	$\#nodes_{TDB}$	$\leq \#nodes_{TDB}$
singletons	$\#nodes_{sing}$	$\leq \#nodes_{sing}$	$\#nodes_{sing}$	$\leq \#nodes_{sing}$
non-singletons	$\#nodes_{ns}$	$\leq \#nodes_{ns}$	0	0

used. More savings were observed when a lower k (e.g., k=1 decimal places) was used for Improvement 1.

6 Conclusions

Most existing algorithms mine frequent patterns from traditional transaction databases that contain precise data. However, there are many real-life situations in which one needs to deal with uncertain data. To handle these situations, we proposed (i) the *UF-tree* to effectively capture the content of transaction databases consisting of uncertain data (in which each item in every transaction is associated with an existential probability) and (ii) a tree-based mining algorithm called *UF-growth* to efficiently find frequent patterns from UF-trees. When compared with U-Apriori, our proposed UF-growth algorithm is superior in performance. In addition, we also presented two improvements (which can be applied independently or simultaneously) to UF-growth. The rounding

of expected support values and the elimination of UF-trees for projected databases for non-singleton patterns both contribute to the reduction of the amount of required memory and further speed-up of the mining process. Hence, with our tree-based approach, users can mine frequent patterns from uncertain data effectively.

Acknowledgement. This project is partially supported by NSERC (Canada) in the form of research grants.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. VLDB, pp. 487–499 (1994)
2. Bonchi, F., Lucchese, C.: Pushing tougher constraints in frequent pattern mining. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 114–124. Springer, Heidelberg (2005)
3. Cheung, W., Zaïane, O.R.: Incremental mining of frequent patterns without candidate generation or support constraint. In: Proc. IDEAS, pp. 111–116 (2003)
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. Dai, X., Yiu, M.L., et al.: Probabilistic spatial queries on existentially uncertain data. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 400–417. Springer, Heidelberg (2005)
6. El-Hajj, M., Zaïane, O.R.: COFI-tree mining: a new approach to pattern growth with reduced candidacy generation. In: Proc. FIMI (2003)
7. Giannella, C., Han, J., et al.: Mining frequent patterns in data streams at multiple time granularities. In: Data Mining: Next Generation Challenges and Future Directions, ch. 6. AAAI/MIT Press (2004)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. SIGMOD, pp. 1–12 (2000)
9. Lakshmanan, L.V.S., Leung, C.K.-S., Ng, R.T.: Efficient dynamic mining of constrained frequent sets. ACM TODS 28(4), 337–389 (2003)
10. Leung, C.K.-S., Carmichael, C.L., Hao, B.: Efficient mining of frequent patterns from uncertain data. In: Proc. IEEE ICDM Workshops, pp. 489–494 (2007)
11. Leung, C.K.-S., Irani, P.P., Carmichael, C.L.: FIsViz: A frequent itemset visualizer. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 644–652. Springer, Heidelberg (2008)
12. Leung, C.K.-S., Khan, Q.I.: DSTree: a tree structure for the mining of frequent sets from data streams. In: Proc. IEEE ICDM, pp. 928–932 (2006)
13. Leung, C.K.-S., Lakshmanan, L.V.S., Ng, R.T.: Exploiting succinct constraints using FP-trees. SIGKDD Explorations 4(1), 40–49 (2002)
14. Ng, R.T., Lakshmanan, L.V.S., et al.: Exploratory mining and pruning optimizations of constrained associations Rules. In: Proc. SIGMOD, pp. 13–24 (1998)
15. Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: Proc. SIGMOD Workshop on DMKD, pp. 21–30 (2000)