Top Down FP-Growth for Association Rule Mining

Ke Wang, Liu Tang, Jiawei Han and Junqiang Liu
School of Computing Science, Simon Fraser University
{wangk, llt, han, jliui}@cs.sfu.ca

Abstract. In this paper, we propose an efficient algorithm, called **TD-FP-Growth** (the shorthand for Top-Down FP-Growth), to mine frequent patterns. **TD-FP-Growth** searches the FP-tree in the top-down order, as opposed to the bottom-up order of previously proposed FP-Growth. The advantage of the top-down search is not generating conditional pattern bases and sub-FP-trees, thus, saving substantial amount of time and space. We extend **TD-FP-Growth** to mine association rules by applying two new pruning strategies: one is to push multiple minimum supports and the other is to push the minimum confidence. Experiments show that these algorithms and strategies are highly effective in reducing the search space.

1 Introduction

Association rule mining has many important applications in real life. An association rule represents an interesting relationship written as $A \Rightarrow B$, read as "if A occurs, then B likely occurs". The probability that both A and B occur is called the *support*, and written as *count*(AB). The probability that B occurs given that A has occurred is called the *confidence*. The association rule mining problem is to find all association rules above the user-specified minimum support and minimum confidence. This is done in two steps: step 1, *find all frequent patterns*; step 2, *generate association rules from frequent patterns*.

This two-step mining approach suffers from several drawbacks. First, only a single uniform minimum support is used, though the distribution of data in reality is not uniform. Second, the two-step process does not consider the confidence constraint at all during the first step. Pushing the confidence constraint into the first step can further reduce search space and hence improve efficiency.

In this paper, we develop a family of algorithms, called **TD-FP-Growth**, for mining frequent patterns and association rules. Instead of exploring the FP-tree in the bottom-up order as in [5], **TD-FP-Growth** explores the FP-tree in the top-down order. The advantage of the top-down search is not constructing conditional pattern bases and sub-trees as in [5]. We then extend **TD-FP-Growth** to mine association rules by applying two new pruning strategies: **TD-FP-Growth(M)** pushes multiple minimum supports and **TD-FP-Growth(C)** pushes the minimum confidence.

2 Related Work

Since its introduction [1], the problem of mining association rules has been the subject of many studies [8][9][10][11]. The most well known method is the Apriori's *anti-monotone* strategy for finding frequent patterns [3]. However, this method suffers from generating too many candidates. To avoid generating many candidates, [4] proposes to represent the database by a *frequent pattern tree* (called the FP-tree). The FP-tree is searched recursively in a bottom-up order to grow longer patterns from shorter ones. This algorithm needs to build conditional pattern bases and sub-FP-trees for each shorter pattern in order to search for longer patterns, thus, becomes very time and space consuming as the recursion goes deep and the number of patterns goes large.

As far as we know, [7][8] are the only works to explicitly deal with non-uniform minimum support. In [7], a minimum item support (MIS) is associated with each item. [8] bins items according to support and specifies the minimum support for combinations of bins. Unlike those works, our specification of minimum support is associated with the consequent of a rule, not with an arbitrary item or pattern.

3 TD-FP-Growth for Frequent Pattern Mining

As in [5], **TD-FP-Growth** first constructs the FP-tree in two scans of the database. In the first scan, we accumulate the count for each item. In the second scan, only the frequent items in each transaction are inserted as a node into the FP-tree. Two transactions share the same upper path if their first few frequent items are same. Each node in the tree is labeled by an item. An *I_node* refers to a node labeled by item *I*. For each item *I*, all *I*_nodes are linked by a *side-link*. Associated with each node *v* is a count, denoted by *count(v)*, representing the number of transactions that pass through the node. At the same time, a header table H(Item, count, side-link) is built. An entry (*I, H(I), ptr*) in the header table records the total count and the head of the side-link for item *I*, denoted by *H*(*I*) and *ptr* respectively. Importantly, the items in each transaction are lexicographically ordered, and so are the labels on each path in FP-tree and the entries in a header table. We use Example 3.1 to illustrate the idea of **TD-FP-Growth**.

Example 3.1 A transaction database is given as in the following Figure 3.1. Suppose that the minimum support is 2. After two scans of transaction database, the FP-tree and the header table H is built as Figure 3.1.

The *top-down mining* of FP-tree is described below. First, entry *a* at the top of H is frequent. Since *a*_node only appears on the first level of the FP-tree, we just need to output {*a*} as a frequent pattern.

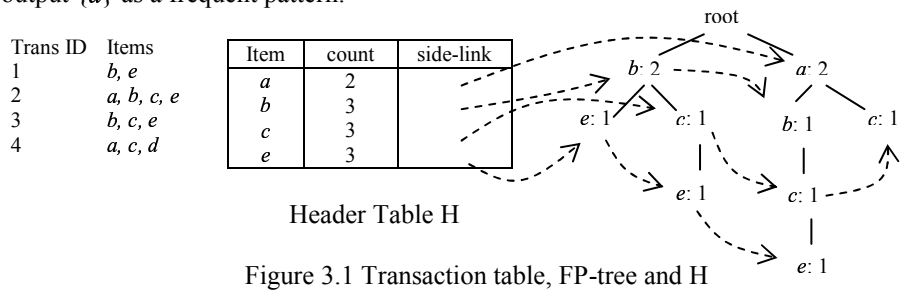

Figure 3.1 Transaction table, FP-tree and H

Then, for entry *b* in H, following the side-link of *b*, we walk up the paths starting from *b*_node in the FP-tree once to build a sub-header-table for *b*, denoted H_*b*. These paths are in bold face in Figure 3.2. During the walk up, we link up encountered nodes of the same label by a side-link, and accumulate the count for such nodes. In Figure 3.2, there are two paths starting with *b*_node: *root-b* and *root-a-b*. By walking up these paths, we accumulate the count of the *a*_node to 1 because path *root-a-b* only occurs once in the database. We also create an entry *a* in the sub-header table H_*b*. The count of entry *a* is 1 since the count of *a* is actually the count of pattern {*a, b*}.

*a*_node is now linked by the side-link for entry *a* in H_*b* and the count is modified to 1. Since the minimum support is 2, pattern {*a, b*} is infrequent. This finishes mining patterns with *b* as the last item. H_*b* now can be deleted from the memory. If pattern {*a, b*} is frequent, we will continue to build sub-header-table H_*ab* and mine FP-tree recursively. In general, when we consider an entry *I* in H_*x*, we will mine all frequent

patterns that end up with *Ix*. In this way, **TD-FP-Growth** finds out the complete set of frequent patterns.

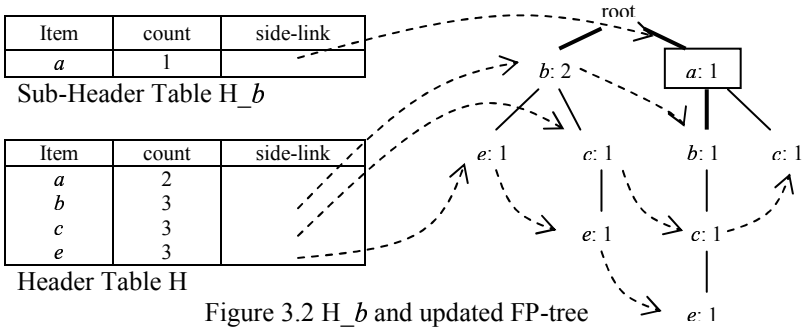

Figure 3.2 H_*b* and updated FP-tree

Similarly, we find frequent patterns: {*c*}, {*b, c*} and {*a, c*} for entry *c,* and {*e*}, {*b, e*}, {*c, e*} and {*b, c, e*} for entry *e*.

Algorithm 1: TD-FP-Growth

Input: a transaction database, with items in each transaction sorted in the lexicographic order, a minimum support: *minsup*. **Output:** frequent patterns above the minimum support. **Method:** build the FP-tree; then call mine-tree *(*∅*, H*);

Procedure mine-tree(*X*, *H*)
(1) **for** each entry *I* (top down order) in *H* **do**
(2) **if** *H*(*I*) >= *minsup*, **then**
(3) output *IX*;
(4) create a new header table H_I by call buildsubtable(*I*);
(5) mine-tree(*IX*, H_I);

Procedure buildsubtable(*I*)
(1) **for** each node *u* on the side-link of *I* **do**
(2) walk up the path from *u* once **do if** encounter a *J*_node *v* **then**
(3) link *v* into the side-link of *J* in H_I;
(4) count(*v*) = count(*v*) + count(*u*);
(5) $H_I(J) = H_I(J)$ + count(*u*);

Unlike **FP-Growth**, **TD-FP-Growth** processes nodes at upper levels before processing those at lower levels. This is important to ensure that any modification made at upper levels would not affect lower levels. Indeed, as we process a lower level node, all its ancestor nodes have already been processed. Thus, to obtain the "conditional pattern base" of a pattern (as named in [5]), we simply walk up the paths above the nodes on the current side-link and update the counts on the paths. In this way, we update the count information on these paths "in place" without creating a copy of such paths. As a result, during the whole mining process, no additional conditional pattern bases and sub-trees are built. This turns out to be a big advantage over the bottom-up **FP-Growth** that has to build a conditional pattern base and sub-tree for each pattern found. Our experiments confirm this performance gain.

4 TD-FP-Growth for Association Rule Mining

In this section, we extend **TD-FP-Growth** for frequent pattern mining to association rule mining. We consider two new strategies for association rule mining problem: push multiple minimum supports and push the confidence constraint.

In the discussion below, we assume that items are divided into *class items* and *non-class items*. Each transaction contains exactly one class item and several non-class items. We consider only rules with one class item on the right-hand side. Class items are denoted by $C_1, ..., C_m$.

4.1 TD-FP-Growth(M) for multiple minimum supports

In a *ToyotaSale* database, suppose that *Avalon* appears in fewer transactions, say 10%, whereas *Corolla* appears in more transactions, say 50%. If a large uniform minimum support, such as 40%, is used, the information about *Avalon* will be lost. On the contrary, if a small uniform minimum support, such as 8%, is used, far too many uninteresting rules about *Corolla* will be found. A solution is to adopt different minimum supports for rules of different classes.

We adopt **TD-FP-Growth** to mine association rules using non-uniform minimum supports. The input to the algorithm is one $minsup_i$ for each class C_i, and the minimum confidence *minconf*. The output is all association rules $X \Rightarrow C_i$ satisfying $minsup_i$ and *minconf*. The algorithm is essentially **TD-FP-Growth** with the following differences. (1) We assume that the class item C_i in each transaction is the last item in the transaction. (2) Every frequent pattern XC_i must contain exactly one class item C_1, where X contains no class item. The support of X is represented by $H(I)$ and the support of XC_i is represented by $H(i, I)$. (3) The minimum support for XC_i is $minsup_i$. (4) We prune XC_i immediately if its confidence computed by by $H(i, I)/H(I)$ is below the minimum confidence.

4.2 TD-FP-Growth(C) for confidence pruning

A nice property of the minimum support constraint is the *anti-monotonicity:* if a pattern is not frequent, its supersets are not frequent either. All existing frequent pattern mining algorithms have used this property to prune infrequent patterns. However, the minimum confidence constraint does not have a similar property. This is the main reason that most association rule mining algorithms ignore the confidence requirement in the step of finding frequent patterns. However, if we choose a proper definition of support, we can still push the confidence requirement inside the search of patterns. Let us consider the following modified notion of support.

Definition 4.1: The *(modified) support* of a rule $A \Rightarrow B$ is $count(A)$.

We rewrite the minimum confidence constraint C: $count(AB)/count(A) >= minconf$ to $count(AB) >= count(A) * minconf$. From the support requirement $count(A) >= minsup$, we have a new constraint C': $count(AB) >= minsup * minconf$. Notice that C' is anti-monotone with respect to the left-hand side A. Also, C' is not satisfied, neither is C. This gives rise to the following pruning strategy.

Theorem 4.1: (1) If $A \Rightarrow B$ does not satisfy C', $A \Rightarrow B$ does not satisfy the minimum confidence either. (2) If $A \Rightarrow B$ does not satisfy C', no rule $AX \Rightarrow B$ satisfies the minimum confidence, where X is any set of items.

Now, we can use both the minimum support constraint and the new constraint C' to prune the search space: if a rule $A \Rightarrow B$ fails to satisfy both the minimum support and C', we prune all rules $AX \Rightarrow B$ for any set of items X. In this way, the search space is tightened up by intersecting the search spaces of the two constraints.

5 Experiment Results

All experiments are performed on a 550MHz AMD PC with 512MB main memory, running on Microsoft Windows NT4.0. All programs are written in Microsoft Visual C++ 6.0. We choose several data sets from UC_Irvine Machine Learning Database Repository: http://www.ics.uci.edu/~mlearn/MLRepository.html.

dataset	# of trans	# of items per trans	class distribution	# of distinct items
Dna-train	2000	61	C_1=23.2%, C_2=24.25%, C_3= 52.55%	240
Connect-4	67557	43	C_1=65.83%, C_2= 24.62%, C_3= 9.55%	126
Forest	581012	13	C_1= 36.36%, C_2= 48.76%, C_3= 6.15%, C_4: 0.47%, C_5= 1.63%, C_6= 2.99%, C_7= 3.53%	15916

Table 5.1 data sets table

Table 5.1 shows the properties for the three data sets. C*onnect-4* is the densest, meaning that there are a lot of long frequent patterns. *Dna-train* comes the next in density. *Forest* is relatively sparse comparing with the other two data sets, however, it is a large data set with 581012 transactions.

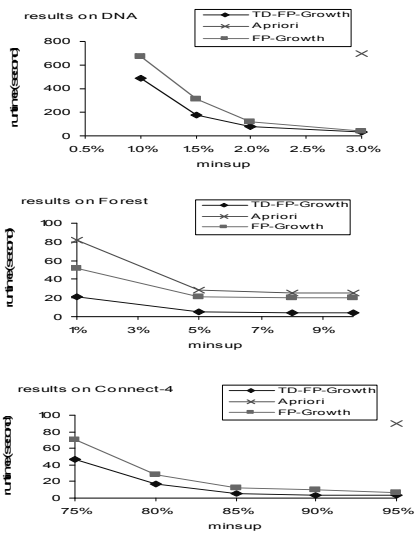

Figure 5.1 frequent pattern mining

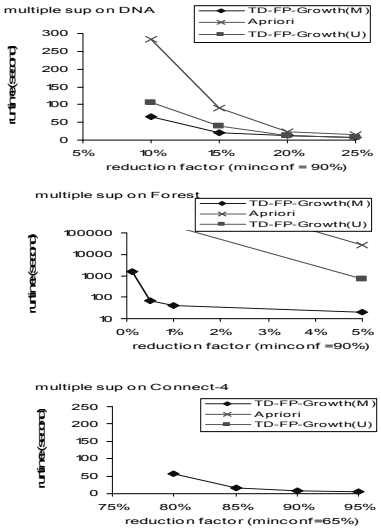

Figure 5.2 multiple support mining

5.1 Frequent pattern mining

In this experiment, we evaluate the performance gain of **TD-FP-Growth** on mining frequent patterns. We compare it with **Apriori** and **FP-Growth**. Figure 5.1 shows a set of curves on the scalability with respect to different minimum supports (minsup). These experiments show that **TD-FP-Growth** is the most efficient algorithm for all data sets and minimum supports tested.

5.2 TD-FP-Growth for multiple minimum supports

In this experiment, we evaluate the performance gain of using multiple minimum supports. We compare three algorithms: **TD-FP-Growth(M**), **TD-FP-Growth(U)**, and **Apriori**. **TD-FP-Growth(U)** is **TD-FP-Growth(M)** in the special case that there is only a single minimum support equal to the smallest minimum supports specified. To specify the minimum support for each class, we multiply a *reduction factor* to the percentage of

each class. For example, if there are two classes in a data set and the percentage of them is 40% and 60%, with the reduction factor of 0.1, the minimum supports for the two classes are 4% and 6% respectively. And the uniform minimum support is set to 4%.

As shown in Figure 5.2, **TD-FP-Growth(M)** scales best for all data sets. Take C*onnect-4* as the example. As we try to mine rules for the rare class *draw*, which occurs in 9.55% of the transactions, the uniform minimum support at reduction factor of 0.95 is 9.55% * 95%, **TD-FP-Growth(U)** ran for more than 6 hours and generated more 115 million rules. However, with the same reduction factor, **TD-FP-Growth(M)** generated only 2051 rules and took only 5 seconds. That's why in Figure 5.2 for C*onnect-4*, the run time for the two algorithms with uniform minimum support is not included.

5.3 TD-FP-Growth for confidence pruning

In this experiment, we evaluate the effectiveness of confidence pruning. The results are reported in Figure 5.3. **TD-FP-Growth(NC)** is **TD-FP-Growth(C)** with confidence pruning turned off. We fix the minimum support and vary the minimum confidence. As shown in Figure 5.3, **TD-FP-Growth(C)** is more efficient than **Apriori** and **TD-FP-Growth(NC)**, for all data sets, minimum supports, and minimum confidences tested.

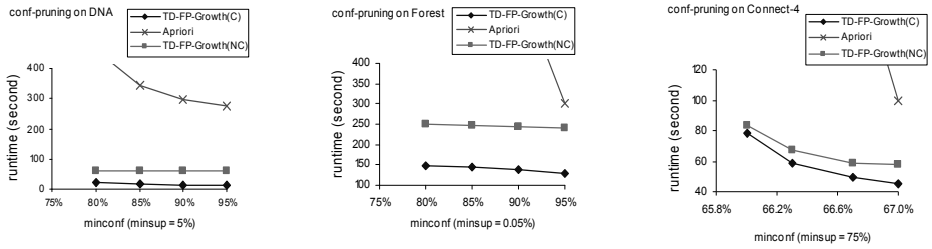

Figure 5.3 confidence-pruning results—time comparison

6 Conclusion

In this paper, we proposed an efficient algorithm for frequent pattern mining, called **TD-FP-Growth**. The main advantage of **TD-FP-Growth** is not building conditional pattern bases and conditional FP-tree as the previously proposed **FP-Growth** does. We extended this algorithm to mine association rules by applying two new pruning strategies. We studied the effectiveness of **TD-FP-Growth** and the new pruning strategies on several representative data sets. The experiments show that the proposed algorithms and strategies are highly effective and outperform the previously proposed **FP-Growth**.

Reference

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *ACM-SIGMOD* 1993, 207-216.

[2] H. Toivonen. Sampling large databases for association rules. VLDB 1996, 134-145.

[3] R. Agrawal and S. Srikant. Mining sequential patterns. ICDE 1995, 3-14.

[4] J. Han, J. Pei and Y. Yin. Mining Frequent patterns without candidate generation. SIGMOD 2000, 1-12.

[5] R. Srikant and R. Agrawal. Mining generalized association rules. VLDB 1995, 407-419.

[6] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. VLDB 1996, 134-145.

[7] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. KDD 1999, 337-341

[8] K. Wang, Y. He and J. Han. Mining frequent patterns using support constraints. VLDB 2000, 43-52.