Journal of Southeast University (English Edition) Vol. 22, No. 1, pp. 48 – 53 Mar. 2006 ISSN 1003—7985

Mining condensed frequent subtree base

Wang Tao Lu Yansheng

(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: In frequent tree pattern mining, the number of frequent subtrees generated is often too large. To tackle this problem, the concept of condensed frequent subtree base is proposed. The base consists of the maximal frequent subtrees for a series of support thresholds. It is a subset of frequent subtrees, and is used to approximate the support of arbitrary frequent subtrees with guaranteed maximal error bound. In addition, an algorithm is developed to mine such a condensed subtree base in a database of labeled rooted ordered trees. The algorithm adopts the way of right-most extension to generate systematically all frequent rooted ordered subtrees. Several techniques are proposed to prune the branches that do not correspond to the maximal frequent subtrees. Heuristic techniques are used to arrange the order of computation so that relatively expensive computation is avoided as much as possible. Experimental results show that the size of the base is less than 10% of that of the complete set, and the algorithm outperforms the previous algorithms.

Key words: data mining; tree pattern; condensed subtree base

Recently, there has been growing interest in mining databases of trees. In the database area, XML documents are often rooted trees where vertices represent elements or attributes and edges represent element-subelement and attribute-value relationships; in Web traffic mining, access trees are used to represent the access patterns of different users[1]. In this paper, we study one important issue in mining databases of labeled rooted ordered trees—finding frequently occurring subtrees.

In Ref. [1], Zaki presented an apriori-like algorithm, TreeMiner, to discover all frequent embedded subtrees (i. e., those subtrees that preserve ancestor-descendant relationships) in a forest or a database of rooted ordered trees. In the apriori-like algorithm, a candidate subtree is generated by joining two frequent subtrees with smaller sizes. In Ref. [2], Asai et al. presented an algorithm, FREQT, to discover frequent rooted ordered subtrees. In the algorithm, a candidate subtree is generated by extending its unique parent, which is a frequent subtree of smaller size.

However, because of the combinatorial explosion, the number of frequent subtrees usually grow exponentially with the tree size. Two consequences result from this exponential growth. First, the end-users will be overwhelmed by the huge number of frequent subtrees presented to them and therefore have difficulty in gaining insights from the frequent subtrees. Second, mining algorithms may become intractable due to the exponential number of frequent subtrees.

In Ref. [3], the concept of a condensed frequent pattern base is proposed and can lead to an error-bound approximation of frequencies of itemsets. In this paper, we generalize the concept in tree pattern mining and propose the notion of a condensed frequent subtree base with guaranteed maximal error bound. Condensing a frequent subtree base leads to more effective frequent subtree mining. By computing a condensed subtree base, the number of subtrees can be reduced dramatically, but the general information about frequent subtrees still retains. A much smaller base of subtrees certainly helps users comprehend the mining results. Computing a much smaller subtree base also leads to better efficiency.

We develop an efficient algorithm to mine such a condensed subtree base from tree database directly. Our results show that computing a condensed frequent subtree base is promising.

1 Problem Definition

A tree is an acyclic connected graph[4]. In this paper, we focus on ordered labeled rooted trees. A tree is denoted as $t(v_0, N, L, E)$, where ① $v_0 \in N$ is the root node; ② N is the set of nodes; ③ L is the set of labels of nodes, for any node $u \in N$, $L(u)$ is the label of u; ④ E is the set of edges in the tree. Please note that two nodes in a tree may carry the identical label.

If (u, v) is an edge in the tree, then u is the parent of v and v is a child of u. For nodes u, v_1 and v_2, if u is

Received 2005-03-22.

Biographies: Wang Tao(1969—), female, graduate; Lu Yansheng (corresponding author), male, professor, lys@mail.hust.edu.cn.

the parent of both v_1 and v_2, then v_1 and v_2 are siblings. A node without any child is a leaf node, otherwise, it is an internal node. In general, an internal node may have multiple children. If for each internal node, all the children are ordered, then the tree is an ordered tree.

Given a tree $t(v_0, N, L, E)$, tree $t'(v_0', N', L', E')$ is called a subtree of t, denoted as $t' \subseteq t$, if ① $N' \subseteq N$; ② for any node $u \in N'$, $L(u) = L'(u)$; ③ $E' \subseteq E$. If t' is a subtree of t, then t is called a supertree of t'.

Let D denote a database where each transaction $s \in D$ is a labeled rooted ordered tree. For a given pattern t (where t is a labeled rooted ordered tree), we say t occurs in a transaction s if t is a subtree of s. Let $\sigma_t(s) = 1$ if t is a subtree of s, and 0 otherwise. We say s supports pattern t if $\sigma_t(s)$ is 1 and we define the support of a pattern t in the database D as $\sup(t) = \sum_{s \in D} \sigma_t(s)$. A pattern t is called a frequent subtree if its support is greater than or equal to a minimum support threshold min_sup specified by a user. The frequent subtree mining problem is to find all frequent subtrees in a given database. The set of all frequent subtrees is called a subtree base, or ST_base in short.

It is often expensive to find the complete set of frequent subtrees, since an ST_base may contain a huge number of frequent subtrees. In this paper, we propose to overcome the difficulty caused by "huge amount of frequent subtrees" as follows: we compute a smaller set of frequent subtrees, i. e., a "condensed ST_base", and then use it to approximate the supports of arbitrary frequent subtrees.

Problem statement Given a tree database, a support threshold, and a user-specified error bound k, the problem of computing a condensed ST_base is to find a subset of frequent subtrees β and a function $\int\beta$ such that the following holds for each subtree t:

$$\int\beta(t) = \begin{cases} 0 & \text{if } t \text{ is infrequent} \\ [\sup_{lb}, \sup_{ub}] & \text{s. t. } \sup_{lb} \leqslant \sup(t) \leqslant \sup_{ub} \\ & \text{and } \sup_{ub} - \sup_{lb} \leqslant k \text{ if } t \text{ is frequent} \end{cases}$$

The function $\int\beta$ is called a (support) approximation function, and the set β is a condensed ST_base w. r. t. $\int\beta$.

2 Constructing a Condensed ST_Base Using Maximal Rooted Ordered Subtrees

Intuitively, we are going to construct a condensed ST_base consisting of maximal frequent subtrees for a series of support thresholds. More specifically, given a support threshold min_sup and error bound k, we divide the set of frequent subtrees into a number of disjoint subsets: ① The set of subtrees with support in the range [min_sup, min_sup + k], ② Those with support in the range [min_sup + k + 1, min_sup + 2k + 1], etc. The i-th subset contains those subtrees with support in the range [min_sup + (i − 1)(k + 1), min_sup + ik + i − 1] where $1 \leqslant i \leqslant (|D| + 1 - \text{min\_sup})/(k+1)$.

Given a frequent subtree, we can approximate its support with the maximal error of k, by determining which subset the subtree belongs to. To determine which subset a subtree belongs to, we only need to record the maximal subtrees at various layers w. r. t. the lower bounds of supports of the ranges.

We now generalize the ideas by providing the definition of a condensed ST_base.

Definition 1 Maximal frequent subtree

A frequent subtree t is a maximal frequent subtree if none of its supertrees is frequent.

Definition 2 Given a database of labeled rooted ordered trees D, support threshold min_sup and error bound k, let the number of levels be

$$n\_\text{level} = \left\lfloor \frac{|D| + 1 - \text{min\_sup}}{k+1} \right\rfloor$$

Define

$$\begin{aligned} &\text{min\_sup}_1 = \text{min\_sup} \\ &\text{min\_sup}_2 = \text{min\_sup} + k + 1 \\ &\vdots \\ &\text{min\_sup}_i = \text{min\_sup} + (i-1)(k+1) \\ &\qquad \text{for } 1 \leqslant i \leqslant n\_\text{level} \end{aligned}$$

Then, $B_m = \bigcup_{i=1}^{i \leqslant n\_\text{level}} M_i$ is called an M_base, w. r. t. the approximation function ξ defined below. Here M_i is the set of maximal subtrees w. r. t. support threshold min_sup_i.

Definition 3 Given an error bound k, an M_base B_m, and a subtree t, let

$$\xi(t) = \begin{cases} 0 & \text{if there exists no } t' \in B_m \text{ s. t. } t' \supseteq t \\ [\sup(t), \sup(t)] & \text{if } t \in B_m \\ [m, m+k] & \text{if } t \notin B_m, \text{ where } m = \\ & \max\{\sup(t') \mid t' \in B_m, t' \supset t\} \end{cases}$$

It can be seen that each M_base B_m is a condensed ST_base w. r. t. function ξ. In essence, for each given subtree t we find the supertree t' of t in B_m having the largest support, and use the range of the support for t' as the estimate of the support of t.

The remaining problem is how to find the maximal subtrees efficiently in the condensed ST_base B_m. The algorithm we adopt proceeds in a manner like

CMTreeMiner proposed in Ref. [5].

A subtree t can be extended to t' by adding a new vertex, the vertex can be at different locations, t is called a parent of t', t' is called a child of t. In this way a subtree can be generated from different parents that will result in redundancies. There is a more clever way to extend subtrees found by Asai et al. [2]. In Asai, et al. 's algorithm, each candidate subtree is generated at most once (and, therefore, redundancies are avoided) from its unique parent. The parent of a subtree is uniquely determined by removing the rightmost vertex of the subtree, where the rightmost vertex of a tree is defined as its last vertex according to the depth-first traversal order. We can systematically generate all frequent rooted ordered subtrees in this way.

Definition 4 Right most extension

The rightmost vertex of a tree t is defined as its last vertex according to the depth-first traversal order. The rightmost path of t is defined as the path from the root to the rightmost vertex of t. When extending a frequent subtree t, an additional vertex w is added as the rightmost child of one vertex on the rightmost path of t (and therefore w becomes the new rightmost vertex). We call this kind of restricted extension as rightmost extension.

Definition 5 For a frequent subtree t, we define the blanket of t w. r. t. a support threshold x, denoted by $B(t, x)$, as the set of immediate supertrees of t whose support is not less than x, where an immediate supertree of t is a supertree t' of t that has one more vertex than t.

Definition 6 For a subtree t and one of its immediate supertrees $t' \in B(t, x)$, we can add a vertex w to t to get t'. We use $t' \setminus t$ to represent the additional vertex w in t' that is not in t. Please notice that $t' \setminus t$ represents not only the vertex label of w, but also its position, i. e., which vertex is w's parent and w's order among its siblings.

Definition 7 For a frequent subtree t and one of its supertrees $t' \in B(t, x)$, the vertex $t' \setminus t$ can be at different locations. According to whether $t' \setminus t$ is the rightmost vertex of t' we divide $B(t, x)$ into two parts: the left-blanket $B_{\text{left}}(t, x)$ and the right-blanket $B_{\text{right}}(t, x)$. For every $t' \in B_{\text{right}}(t, x)$, $t' \setminus t$ is the rightmost vertex of t'. For every $t' \in B_{\text{left}}(t, x)$, $t' \setminus t$ is not the rightmost vertex of t'.

Based on definition 5, we have the following lemma.

Lemma 1 Let t be a frequent subtree and $i = \max\{j \mid \sup(t) \geqslant \min\_\sup_j\}$. Then, t is maximal w. r. t. $\min\_\sup_i$ iff $B(t, \min\_\sup_i) = \varnothing$.

3 Search Space Pruning

The final goal of our algorithm is to find only maximal frequent subtrees at various layers. Therefore, it is not necessary to grow the complete set of frequent subtrees. Can we prune some unpromising subtrees as early as possible? We have the following definitions and theorems.

Definition 8 Occurrence-matching and transaction-matching

If a subtree t occurs in a transaction s, it can occur more than once. We call each of them an occurrence of t in the database. For $t' \in B(t, x)$, we define t' and t as occurrence-matched if for each occurrence of t in (a transaction of) the database, there is at least one (there may be more than one) corresponding occurrence of t'; we say that t' and t are transaction-matched if for each transaction $s \in D$ such that $\sigma_t(s) = 1$, we have $\sigma_{t'}(s) = 1$.

Accordingly, we define the following subsets of $B(t, x)$:

$$B^{\text{OM}}(t, x) = \{t' \in B(t, x) \mid t' \text{ and } t \text{ are occurrence-matched}\}$$

$$B^{\text{TM}}(t, x) = \{t' \in B(t, x) \setminus B^{\text{OM}}(t, x) \mid t' \text{ and } t \text{ are transaction-matched}\}$$

$$B^{\text{F}}(t, x) = B(t, x) \setminus (B^{\text{OM}}(t, x) \cup B^{\text{TM}}(t, x))$$

Lemma 2 For a frequent subtree t, if there exists $t' \in B^{\text{OM}}_{\text{left}}(t, \min\_\sup)$, then ① t is not maximal and ② for each $t'' \in B_{\text{right}}(t, \min\_\sup)$, there exists at least one supertree $t''' \in B^{\text{OM}}_{\text{left}}(t'', \min\_\sup)$.

Proof The first statement is true because $\sup(t) = \sup(t')$. For the second statement, we notice that $t' \setminus t$ occurs at each occurrence of t so it occurs at each occurrence of t''; in addition, $t' \setminus t$ is on the left of the rightmost path of t so $t' \setminus t$ will never occur in t''. Therefore, we can obtain the t''' that satisfies the requirement by adding $t' \setminus t$ to t''.

Theorem 1 For a frequent subtree t, if there exists $t' \in B^{\text{OM}}_{\text{left}}(t, \min\_\sup)$, then neither t nor any t's descendants can be maximal, and therefore t (together with all t's descendants) can be pruned from the search space.

Proof By inductively applying lemma 2 to t and its children(for each $t'' \in B_{\text{right}}(t, \min\_\sup)$), we have the claim.

Theorem 2 For a frequent subtree t, if there exists $t' \in B^{\text{OM}}_{\text{right}}(t, \min\_\sup)$, and the parent of $t' \setminus t$ is v (where v is a vertex on the rightmost path of t), then we do not have to extend t by adding new rightmost

vertices to any proper ancestor of v.

Proof Assume the preconditions in the theorem hold, and there is a child t'' of t that is obtained by adding a new rightmost vertex $t''\backslash t$ to t, where the parent v' of $t''\backslash t$ is a proper ancestor of v. Because v' is a proper ancestor of v, v is on the left of the rightmost path of t''; in addition, because $t'\backslash t$ is a child of v, $t'\backslash t$ is also on the left of the rightmost path of t''. As a result we can construct a $t''' \in B^{\mathrm{OM}}_{\mathrm{left}}(t'', \min\_\sup)$ by adding $t'\backslash t$ to t'' and obviously t''' and t'' are occurrence-matched. Therefore by theorem 1, neither t'' nor its descendants can be maximal.

4 Order of Computation

To determine if a frequent subtree t is maximal w. r. t. $\min\_\sup_i (i = \max\{j \mid \sup(t) \geqslant \min\_\sup_j\})$, we will have to compute $B(t, \min\_\sup_i)$; to determine if a frequent subtree t can be pruned from the search space, we will have to compute $B^{\mathrm{OM}}(t, \min\_\sup)$. As we know, $B(t, \min\_\sup_i) = B^{\mathrm{OM}}(t, \min\_\sup_i) \cup B^{\mathrm{TM}}(t, \min\_\sup_i) \cup B^{\mathrm{F}}(t, \min\_\sup_i)$.

Based on definition 8, we have the following lemma.

Lemma 3 For a frequent subtree t, $i = \max\{j \mid \sup(t) \geqslant \min\_\sup_j\}$, then $B^{\mathrm{OM}}(t, \min\_\sup_i) = B^{\mathrm{OM}}(t, \min\_\sup)$ and $B^{\mathrm{TM}}(t, \min\_\sup_i) = B^{\mathrm{TM}}(t, \min\_\sup)$.

Based on definition 8 and lemma 3, we know that $B(t, \min\_\sup_i)$ has three different subsets and $B^{\mathrm{OM}}(t, \min\_\sup)$ is one of these three subsets. Now we study in detail how to compute each of these three subsets and show that their computational costs are different.

1) Compute $B^{\mathrm{OM}}(t, \min\_\sup)$ i. e. $B^{\mathrm{OM}}(t, \min\_\sup_i)$

For a frequent subtree t, each occurrence of t has some candidate supertrees for $B^{\mathrm{OM}}(t, \min\_\sup)$. To finally determine $B^{\mathrm{OM}}(t, \min\_\sup)$, we compute the intersection of these candidate supertrees from each occurrence of t, because for a supertree $t' \in B^{\mathrm{OM}}(t, \min\_\sup)$, $t'\backslash t$ should occur with each occurrence of t. We see that computing $B^{\mathrm{OM}}(t, \min\_\sup)$ is not an expensive operation: $B^{\mathrm{OM}}(t, \min\_\sup)$ is just the intersection of the candidate supertrees from all occurrences of t.

2) Compute $B^{\mathrm{TM}}(t, \min\_\sup)$ i. e. $B^{\mathrm{TM}}(t, \min\_\sup_i)$

Computing $B^{\mathrm{TM}}(t, \min\_\sup)$ is similar to computing $B^{\mathrm{OM}}(t, \min\_\sup)$, except that each transaction, instead of each occurrence, has its candidate supertrees for $B^{\mathrm{TM}}(t, \min\_\sup)$. If there are multiple occurrences of t in the same transaction, then the union of the candidate supertrees from these occurrences is taken as the candidate supertrees for the transaction. Again, we can see that computing $B^{\mathrm{TM}}(t, \min\_\sup)$ is not an expensive operation, although it is slightly more expensive than computing $B^{\mathrm{OM}}(t, \min\_\sup)$. In addition, we can see that the computations of $B^{\mathrm{OM}}(t, \min\_\sup)$ and $B^{\mathrm{TM}}(t, \min\_\sup)$ do not involve storing any support, and the computations may be terminated before visiting all the occurrences of t. Whenever the intersection of candidate supertrees becomes empty, the rest of the occurrences can be skipped.

3) Compute $B^{\mathrm{F}}(t, \min\_\sup_i)$

Computing $B^{\mathrm{F}}(t, \min\_\sup_i)$, however, is relatively expensive for the following reason. Again, each transaction has candidate supertrees for $B^{\mathrm{F}}(t, \min\_\sup_i)$, and if there are multiple occurrences of t in a transaction, the union of the candidate supertrees from each of these occurrences is the set of candidate supertrees from the transaction. This time, however, instead of a simple intersection, the union of all the candidate supertrees from each transaction is used. Furthermore, the support of each candidate supertree must be stored and updated in the process, although a large part of the candidate supertrees may ultimately turn out to be infrequent. Therefore, computing $B^{\mathrm{F}}(t, \min\_\sup_i)$ can be relatively expensive.

Based on the above analysis, in order to avoid computing $B^{\mathrm{F}}(t, \min\_\sup_i)$ as much as possible, we adopt the following order to compute the three subsets:

① Compute $B^{\mathrm{OM}}(t, \min\_\sup)$. If $\exists\, t' \in B^{\mathrm{OM}}_{\mathrm{left}}(t, \min\_\sup)$, prune t; else if $\exists\, t' \in B^{\mathrm{OM}}_{\mathrm{right}}(t, \min\_\sup)$, apply theorem 2. In either case, there is no need to compute $B^{\mathrm{TM}}(t, \min\_\sup)$ or $B^{\mathrm{F}}(t, \min\_\sup_i)$, because t is not maximal w. r. t. $\min\_\sup_i$.

② Extend t by the way of right-most extension, get $B_{\mathrm{right}}(t, \min\_\sup)$.

③ Compute $B^{\mathrm{TM}}(t, \min\_\sup)$ if $B^{\mathrm{OM}}(t, \min\_\sup) = \varnothing$. If $B^{\mathrm{TM}}(t, \min\_\sup) \neq \varnothing$, there is no need to compute $B^{\mathrm{F}}(t, \min\_\sup_i)$, because t is not maximal w. r. t. $\min\_\sup_i$.

④ Get $B^{\mathrm{F}}_{\mathrm{right}}(t, \min\_\sup_i)$ if $B^{\mathrm{OM}}(t, \min\_\sup) = \varnothing$ and $B^{\mathrm{TM}}(t, \min\_\sup) = \varnothing$. $B^{\mathrm{F}}_{\mathrm{right}}(t, \min\_\sup_i) = \{t' \mid t' \in B_{\mathrm{right}}(t, \min\_\sup)$ s. t. $\sup(t') \geqslant \min\_\sup_i\}$. If $B^{\mathrm{F}}_{\mathrm{right}}(t, \min\_\sup_i) \neq \varnothing$, then there is no need to compute $B^{\mathrm{F}}_{\mathrm{left}}(t, \min\_\sup_i)$, because t is not maximal w. r. t. $\min\_\sup_i$.

⑤ Compute $B^{\mathrm{F}}_{\mathrm{left}}(t, \min\_\sup_i)$ if $B^{\mathrm{OM}}(t, \min\_$

sup) = ∅, $B^{TM}(t, \min\_sup) = \varnothing$, and $B^{F}_{right}(t, \min\_sup_i) = \varnothing$. If $B^{F}_{left}(t, \min\_sup_i) = \varnothing$, then t is maximal w. r. t. $\min\_sup_i$, otherwise, t is not maximal w. r. t. $\min\_sup_i$.

We summarize the algorithm for constructing an $M_$base as follows.

Algorithm 1 BmMining(D, min_sup, $k\%$)

Input: a database of labeled rooted ordered trees D, the minimum support threshold min_sup, and error bound $k\%$

Output: an $M_$base B_m w. r. t. ξ

$B_m = \varnothing$;

C = frequent 1 − trees(D, min_sup);

BmSpan (C, B_m, D, min_sup);

return B_m.

Algorithm 2 BmSpan(C, B_m, D, min_sup)

Input: a set of frequent 1-trees C, an $M_$base B_m w. r. t. ξ, a database of labeled rooted ordered trees D, and the minimum support threshold min_sup

Output: an $M_$base B_m w. r. t. ξ

```
For each t ∈ C do {
    compute B^OM(t, min_sup);
    if ∃ t' ∈ B^OM_left(t, min_sup) then continue;
    E = ∅;
    for each vertex v on the rightmost path of t do {
        /* (in a bottom-up fashion) */
        for each valid new rightmost vertex w of t do {
        t' ← t plus vertex w, with v as w's parent;
        if sup(t') ≥ min_sup then E = E ∪ {t'};
    }
    if ∃ t' ∈ B^OM_right(t, min_sup) s. t. v is the parent of t' \ t
      then break;
  }
  if E ≠ ∅ then BmSpan(E, B_m, D, min_sup);
  if B^OM(t, min_sup) = ∅ then compute B^TM(t, min_sup);
  if B^OM(t, min_sup) = ∅ and B^TM(t, min_sup) = ∅ then {
    i = max{j | sup(t) ≥ min_sup_j};
    B^F_right(t, min_sup_i) = {t' | t' ∈ E s. t. sup(t') ≥ min_sup_i};
    if B^F_right(t, min_sup_i) = ∅ then {
        compute B^F_left(t, min_sup_i);
        if B^F_left(t, min_sup_i) = ∅ then B_m = B_m ∪ {t}; }
  }
}
```

5 Experiments

To evaluate the effectiveness and efficiency of condensed ST_ bases, we conducted a set of experiments. All the experiments are performed on a 2 GHz Pentium PC machine with 1 GB main memory, running Microsoft Windows 2000. All the programs are coded using Microsoft Visual C++6.0.

The synthetic dataset we used for our experiments is generated using the tree generation program described in Ref. [1]. In brief, a mother tree is generated first with the following parameters: the number of distinct node labels $N = 100$, the total number of nodes in the tree $M = 10^4$, the maximal depth of the tree $D = 10$ and the maximum fanout $F = 10$. The dataset is then generated by creating subtrees of the mother tree. In our experiments, we set the total number of trees in the dataset to be $T = 10^6$. The average number of nodes in each tree is 6.94.

The compression effects of condensed ST_ base can be measured by compression ratio δ defined as follows: δ = (# of subtrees in the condensed ST_ base)/(total # of frequent subtrees). Clearly, the smaller the compression ratio is, the better the compression effect is.

First, we fix the support the threshold and test the compression ratio with respect to various error bounds. The result is shown in Fig. 1. Here, the error bound is set as a percentage of the total number of transactions in the dataset. As can be seen, the larger the error bound is, the better the compression ratio is.

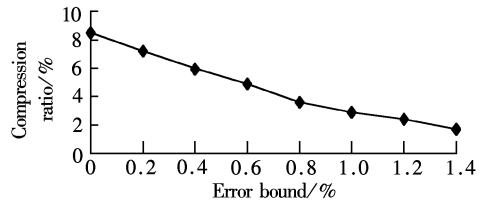

Fig. 1 Compression ratio of B_m w. r. t. error bound (min_sup = 0.01%)

In order to test the efficiency of computing condensed ST_ bases, we compare the runtime of FREQT[6] and BmSpan with respect to the support threshold in Fig. 2. The error bound is set to 1%. The rate of increase in runtime for BmSpan is much slower than that for FREQT as the support threshold decreases. And when the support threshold is low, the trends in runtime of BmSpan and FREQT become more distinct.

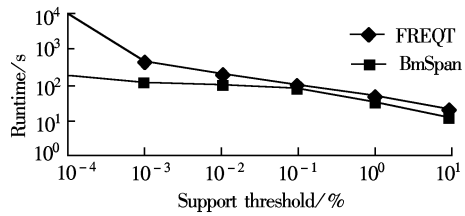

Fig. 2 Runtime w. r. t. support threshold (error bound = 1%)

In addition, to check the effect of the heuristic order of computation that is introduced in section 4, we reverse the order of computation such that $B^F(t, \min\_sup_i)$ is always computed first. This version of

BmSpan is called BmSpan_x. We compare the runtime of BmSpan_x and BmSpan with respect to the support threshold in Fig. 3. The error bound is set to 1%. As can be seen, BmSpan_x always performs worse than BmSpan.

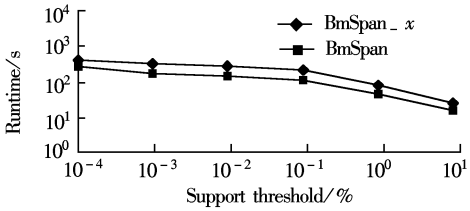

Fig. 3 Runtime w. r. t. support threshold (error bound = 1%)

In summary, condensed ST_bases can achieve non-trivial compression for frequent subtrees; the larger the error bound is, the more we compress. Error bound can help to make the condensed ST_bases more compact. BmSpan is much faster than FREQT, so BmSpan is an efficient algorithm for computing a condensed ST_base.

6 Conclusion

In this paper, we introduce and consider the problem of mining a condensed frequent subtree base from databases of rooted labeled ordered trees. The notion of a condensed ST_base is introduced to significantly reduce the set of subtrees that need to be mined, stored, and analyzed, while providing guaranteed error bound for frequencies of subtrees is not in the base. An algorithm is developed to mine condensed ST_bases. Various pruning and heuristic techniques are used in the algorithm to reduce the search space and to improve the computational efficiency. Experimental results show that we can achieve a substantial compression ratio of condensation using the condensed ST_bases, and our algorithm is efficient. As for future work, it will be interesting to explore other effective condensed ST_bases and efficient mining methods.

References

[1] Zaki M J. Efficiently mining frequent trees in a forest [C]//*8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Alberta, Canada, 2002: 68 - 75.

[2] Asai T, Abe K, Kawasoe S, et al. Efficient substructure discovery from large semi-structured data [C]//*Proc* 2002 *SIAM Int Conf on Data Mining* (*SDM*'02). Arlington, VA, 2002: 457 - 473.

[3] Pei J, Dong G, Zou W, et al. On computing condensed frequent pattern bases [C]//*Proc* 2002 *Int Conf on Data Mining* (*ICDM*'02). Maebashi, Japan, 2002: 378 - 385.

[4] Wang C, Hong M, Pei J, et al. Efficient pattern-growth methods for frequent tree pattern mining [C]//*The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD*'04). Sydney, Australia, 2004: 441 - 451.

[5] Chi Y, Yang Y, Xia Y, et al. CMTreeMiner: mining both closed and maximal frequent subtrees [C]//*The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD*'04). Sydney, Australia, 2004: 63 - 73.

[6] Kudo T. FREQT: an implementation of FREQT [EB/OL]. (2003-03-21) [2004-12-15]. http://chasen.org/~taku/software/freqt/.

挖掘频繁子树精简基

王 涛 卢炎生

(华中科技大学计算机科学与技术学院,武汉 430074)

摘要: 为了解决频繁树模式挖掘中频繁子树的数目通常太大的问题,提出了频繁子树精简基的概念,精简基由相对于一系列支持度阈值的最大频繁子树组成,它是频繁子树的一个子集,可用来估计任一频繁子树的支持度,并能将误差控制在确定范围内. 提出了一个在带标号的有根的有序树的数据库中挖掘这种子树精简基的算法,该算法采用最右扩展方法系统地生成所有的频繁有序有根子树. 采用的剪枝技术能尽早地剪掉一些不可能生成最大频繁子树的分枝,还采用了启发式的技术来安排计算的次序以尽可能避免代价高的计算. 实验结果表明该精简基的大小不到全集的 10%,算法的性能也比挖掘全集的算法要高.

关键词: 数据挖掘;树模式;子树精简基

中图分类号: TP311