Unordered Tree Mining with Applications to Phylogeny

Dennis Shasha*
Courant Institute of Mathematical Sciences
New York University
New York, NY 10012, USA

Jason T. L. Wang† Sen Zhang
College of Computing Sciences
New Jersey Institute of Technology
University Heights, Newark, NJ 07102, USA

Abstract

Frequent structure mining (FSM) aims to discover and extract patterns frequently occurring in structural data, such as trees and graphs. FSM finds many applications in bioinformatics, XML processing, Web log analysis, and so on. In this paper we present a new FSM technique for finding patterns in rooted unordered labeled trees. The patterns of interest are cousin pairs in these trees. A cousin pair is a pair of nodes sharing the same parent, the same grandparent, or the same great-grandparent, etc. Given a tree T, our algorithm finds all interesting cousin pairs of T in $O(|T|^2)$ time where $|T|$ is the number of nodes in T. Experimental results on synthetic data and phylogenies show the scalability and effectiveness of the proposed technique. To demonstrate the usefulness of our approach, we discuss its applications to locating co-occurring patterns in multiple evolutionary trees, evaluating the consensus of equally parsimonious trees, and finding kernel trees of groups of phylogenies. We also describe extensions of our algorithms for undirected acyclic graphs (or free trees).

1 Introduction

Frequent structure mining (FSM) is a problem of continuing interest [22, 45, 48]. The goal of FSM is to discover and extract patterns frequently occurring in tree and graph structures, which often represent complex interactions among entities. FSM finds many applications in bioinformatics, XML processing, scientific data management, Web log analysis, and so on. In this paper we present a new FSM technique for finding patterns in rooted unordered labeled trees. A rooted unordered labeled tree is a tree in which there is a root for the tree, each node may have a label, and the left-to-right order among siblings is unimportant.[1]

The patterns we want to find from these trees contain "cousin pairs." For example, consider the three trees in Figure 1. In the figure, a and y are cousins with distance 0 in T_1; e and f are cousins with distance 0.5 in T_2; b and f are cousins with distance 1 in all the three trees.

The measure "distance" represents kinship of two nodes; two cousins with distance 0 are siblings, sharing the same parent node. Cousins of distance 1 share the same grandparent. Cousins of distance 0.5 represent aunt-niece relationships. Our algorithms can find cousins of varying distances in a single tree or multiple trees.

The cousins (patterns) discovered from trees can be used in several ways. As we will show later, they can be used to evaluate the quality of a consensus tree [32] obtained from multiple phylogenies or can be used to compute the distance between two phylogenies.

Scientists model the evolutionary history of a set of taxa (organisms or species) that have a common ancestor using rooted unordered labeled trees, also known as phylogenetic trees (phylogenies) or evolutionary trees. The internal nodes within a particular tree represent older organisms from which their child nodes descend. The children represent divergences in the genetic composition in the parent organism. Since these divergences cause new organisms to evolve, these organisms are shown as children of the previous organism. Cousin pairs in these trees represent evolutionary relationships between species that share a common ancestor. Finding the cousin pairs helps to better understand the evolutionary history of the species. Later in the paper we also discuss extensions of our techniques for undirected acyclic graphs (or free trees).

1.1 Related Work

In the past, much work on frequent structure mining was conducted with applications to XML, document and semi-

*Work of this author was supported in part by NSF grants IIS-9988345, MCB-0209754 and by NIH grant GM32877 (email: shasha@cs.nyu.edu).

†Work of this author was supported in part by NSF grant IIS-9988636 (email: wangj@njit.edu).

[1] We shall refer to rooted unordered labeled trees simply as trees when the context is clear.

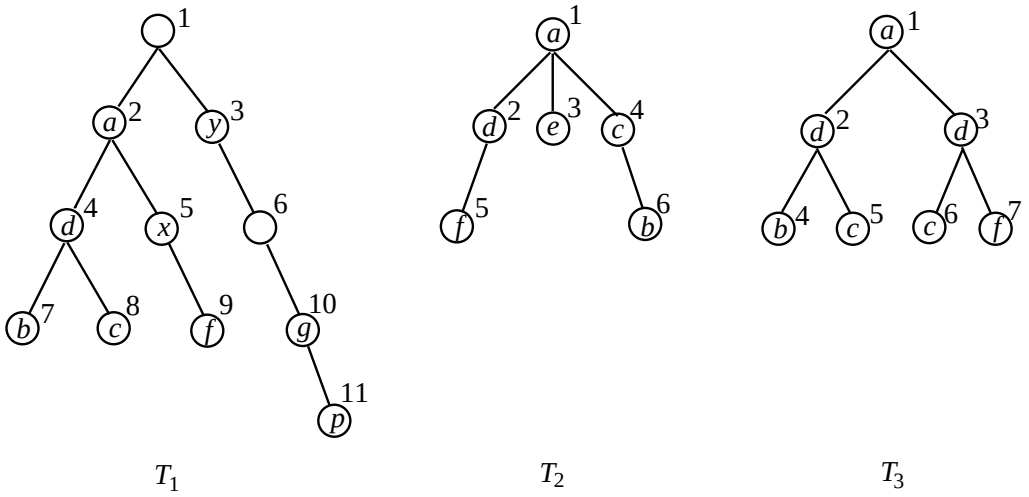

Figure 1. Three trees T_1, T_2 and T_3. Each node in a tree may or may not have a label, and is associated with a unique identification number (represented by the integer outside the node).

structured data processing [3, 25, 28, 29, 42, 43, 44]. The major difference between these works lies in the different patterns they discover, which range from XML DTD, tags, schemes, to structural associations in the documents. More recently Zaki [48] developed algorithms capable of finding frequent embedded tree patterns in a forest where he models a document by an ordered labeled tree. Chen *et al.* [6] studied techniques for selectivity estimation in the context of XML querying. Other related work on general tree matching, inclusion and isomorphism detection can be found in [7, 21, 35].

There has also been work in graph mining. For example, the authors of [20, 22] extended the Apriori technique [2], originally designed for association mining, to find frequent subgraphs in graph data. Yan and Han [46] found closed frequent subgraphs in the graph data. Dehaspe *et al.* [10] applied inductive logic programming to find frequent substructures (subgraphs) describing the carcinogenesis of chemical compounds. Cook and Holder [8] found repeated patterns in graphs using the minimum description length principle. Yoshida and Motoda [47] used beam search for mining subgraphs. Wang *et al.* [45] applied geometric hashing techniques to find frequent substructures in 3-D graphs and used the substructures to classify proteins and chemical compounds.

Our cousin-based distance measure joins the many others already developed [15, 27, 33, 36]. Our work differs from the above approaches in three ways. First, in contrast to the other tree mining methods (e.g. [40, 48]), which focused on trees where the order of siblings matters, we are concerned with unordered trees, as is appropriate for the phylogenetic application. Second, our algorithms find cousin pairs with varying distances from the trees. These frequent cousin pairs differ from the patterns found in all the previous work, entailing new methods in the discovery process. When applied to phylogeny, the proposed cousin-based distance measure can be used to compare evolutionary trees for which existing methods are not suitable. Third, in contrast to the other graph mining methods (e.g. [8, 47]) which are based on a heuristic search and hence may miss some interesting patterns, we perform a complete search without missing any patterns satisfying the user-specified requirement.

The rest of the paper is organized as follows. Section 2 introduces notation and terminology. Section 3 presents algorithms for finding frequent cousin pairs in trees. Section 4 reports experimental results on both synthetic data and phylogenies, showing the scalability and effectiveness of the proposed approach. Section 5 discusses applications of our approach to locating co-occurring patterns in multiple phylogenies, evaluating the consensus of equally parsimonious trees, and finding kernel trees from groups of phylogenies. Section 6 describes extensions of our algorithms for undirected acyclic graphs. Section 7 concludes the paper and points out some future work.

2 Preliminaries

Let Σ be a finite set of labels. A rooted unordered labeled tree of size $k > 0$ on Σ is a quadruple $T = (V, N, L, E)$, where

- V is the set of nodes of T in which a node $r(T) \in V$

is designated as the root of T and $|V| = k$.

- $N : V \mapsto \{1 \ldots, k\}$ is a numbering function that assigns a unique identification number $N(v)$ to each node $v \in V$.

- $L : V' \mapsto \Sigma$, $V' \subseteq V$, is a labeling function that assigns a label $L(v)$ to each node $v \in V'$; the nodes in $V - V'$ do not have a label. Obviously, this labeling function allows multiple nodes to have the same label.

- $E \subset N(V) \times N(V)$ contains all parent-child pairs in T.

For example, refer to the trees in Figure 1. The node numbered 6 in T_1 does not have a label. The nodes numbered 2, 3 in T_3 have the same label d and the nodes numbered 5, 6 in T_3 have the same label c. We now introduce a series of definitions that will be used in our algorithms.

Cousin distance Given two labeled nodes u, v of tree T where neither node is the parent of the other, we represent the least common ancestor, w, of u and v as $lca(u, v)$, and represent the height of u, v respectively, in the subtree rooted at w as $height(u, w)$, $height(v, w)$ respectively. We define the *cousin distance* of u and v, denoted $c\_dist(u, v)$, as shown in Figure 2. The cousin distance $c\_dist(u, v)$ is undefined if $|height(u, w) - height(v, w)|$ is greater than 1, or one of the nodes u, v is unlabeled. (The cutoff of 1 is a heuristic choice that works well for phylogeny. In general there could be no cutoff or the cutoff could be much greater.)

Our cousin distance definition is inspired by genealogy [16]. Node u is a first cousin of v, or $c\_dist(u, v)$ = 1, if u and v share the same grandparent. In other words, v is a child of u's aunts or vice versa. Node u is a second cousin of v, or $c\_dist(u, v)$ = 2, if u and v have the same great-grandparent, but not the same grandparent. For two nodes u, v that are siblings, i.e. they share the same parent, $c\_dist(u, v)$ = 0.

We use the number "0.5" to represent the "once removed" relationship. When the word "removed" is used to describe a relationship between two nodes, it indicates that the two nodes are from different generations. The words "once removed" mean that there is a difference of one generation. For any two labeled nodes u and v, if u is v's parent's first cousin, then u is v's first cousin, once removed [16] and $c\_dist(u, v)$ = 1.5. "Twice removed" means that there is a two-generation difference. Our cousin distance definition requires $|height(u, w) - height(v, w)| \leq 1$ and excludes the twice removed relationship. As mentioned above, this is a heuristic rather than a fundamental restriction.

Distance	Cousin Pair Items
0	$(b, c, 0, 1), (c, f, 0, 1), (d, d, 0, 1)$
0.5	$(d, b, 0.5, 1), (d, c, 0.5, 2), (d, f, 0.5, 1)$
1	$(b, f, 1, 1), (b, c, 1, 1), (c, c, 1, 1),$ $(c, f, 1, 1)$

Table 1. Cousin pair items of T_3 in Figure 1.

For example, consider again T_1 in Figure 1. There is a one-generation difference between the aunt-niece pair y, x and $c\_dist(y, x)$ = 0.5. Node b is node f's first cousin and $c\_dist(b, f)$ = 1. Node d is node g's first cousin, once removed, and $c\_dist(d, g)$ = 1.5. Node f is node g's second cousin, and $c\_dist(f, g)$ = 2. Node f is node p's second cousin, once removed, and $c\_dist(f, p)$ = 2.5.

Notice that parent-child relationships are not included in our work because the internal nodes of phylogenetic trees usually have no labels. So, we do not treat parent-child pairs at all. This heuristic works well in phylogenetic applications, but could be generalized. We proposed one such generalization using the UpDown distance [39]. Another approach, suggested by a reviewer, is to use one upper limit parameter for inter-generational (vertical) distance and another upper limit parameter for horizontal distance.

Cousin pair item Let u, v be cousins in tree T. A *cousin pair item* of T is a quadruple $(L(u), L(v), c\_dist(u, v), occur(u, v))$ where $L(u)$ and $L(v)$ are labels of u, v, respectively, $c\_dist(u, v)$ is the cousin distance of u, v and $occur(u, v) > 0$ is the number of occurrences of the cousin pair in T with the specified cousin distance. Table 1 lists all the cousin pair items of tree T_3 in Figure 1. Consider, for example, the cousin pair item (d, c, 0.5, 2) in the second row of Table 1. Node 2 and node 6, node 3 and node 5 respectively, is an aunt-niece pair with cousin distance 0.5. When taking into account labels of these nodes, we see that the cousin pair (d, c) with distance 0.5 occurs 2 times totally in tree T_3, and hence (d, c, 0.5, 2) is a valid cousin pair item in T_3.

We may also consider the total number of occurrences of the cousins u and v regardless of their distance, for which case we use λ in place of $c\_dist(u, v)$ in the cousin pair item. For example, in Table 1, T_3 has $(b, c, 0, 1)$ and $(b, c, 1, 1)$, and hence we obtain $(b, c, \lambda, 2)$. Here, the cousin pair (b, c) occurs once with distance 0 and occurs once with distance 1. Therefore, when ignoring the distance, the total number of occurrences of (b, c) is 2. Likewise we can ignore the number of occurrences of a cousin pair (u, v) by using λ in place of $occur(u, v)$ in the cousin pair item. For example, in Table 1, T_3 has $(b, c, 0, \lambda)$ and $(b, c, 1, \lambda)$. We may ignore both the cousin distance and the number of occurrences and focus on the cousin

$$c\_dist(u,v) = \begin{cases} height(u,w) - 1 & \text{if } height(u,w) = height(v,w) \\ \max\{height(u,w), height(v,w)\} - 1.5 & \text{if } |height(u,w) - height(v,w)| = 1 \end{cases}$$

Figure 2. Definition of the cousin distance between two nodes u and v.

labels only. For example, T_3 has (b, c, λ, λ), which simply indicates that b, c are cousins in T_3.

Frequent cousin pair Let $\mathcal{S} = \{T_1, T_2, \ldots, T_n\}$ be a set of n trees and let d be a given distance value. We define $\delta_{u,v,i}$ to be 1 if T_i has the cousin pair item $(L(u), L(v), d, occur(u,v))$, $occur(u,v) > 0$; otherwise $\delta_{u,v,i}$ is 0. We define the *support* of the cousin pair (u,v) with respect to the distance value d as $\Sigma_{1 \le i \le n} \delta_{u,v,i}$. Thus the support value represents the number of trees in the set $\mathcal{S}$ that contain *at least* one occurrence of the cousin pair (u,v) having the specified distance value d. A cousin pair is *frequent* if its support value is greater than or equal to a user-specified threshold, *minsup*.

For example, consider Figure 1 again. T_1 has the cousin pair item $(c, f, 1, 1)$, T_2 has the cousin pair item $(c, f, 0.5, 1)$ and T_3 has the cousin pair item $(c, f, 1, 1)$ and $(c, f, 0, 1)$. The support of (c, f) w.r.t. distance 1 is 2 because both T_1 and T_3 have this cousin pair with the specified distance. One can also ignore cousin distances when finding frequent cousin pairs. For example, the support of (c, f) is 3 when the cousin distances are ignored.

Given a set $\mathcal{S}$ of trees, our approach offers the user several alternative kinds of frequent cousin pairs in these trees. For example, the algorithm can find, in a tree T of $\mathcal{S}$, all cousin pairs in T whose distances are less than or equal to *maxdist* and whose occurrence numbers are greater than or equal to *minoccur*, where *maxdist* and *minoccur* are user-specified parameters. The algorithm can also find all frequent cousin pairs in $\mathcal{S}$ whose distance values are at most *maxdist* and whose support values are at least *minsup* for a user-specified *minsup* value. In the following section, we will describe the techniques used in finding these frequent cousin pairs in a single tree or multiple trees.

3 Tree Mining Algorithms

Given a tree T and a node u of T, let $children\_set(u)$ contain all children of u. Our algorithm preprocesses T to obtain $children\_set(u)$ for every node u in T. We also preprocess T to be able to locate a list of all ancestors of any node u in $O(1)$ time using a conventional hash table.

Now, given a user-specified value $maxdist$, we consider all valid distance values 0, 0.5, 1, 1.5, ..., $maxdist$. For each valid distance value d, we define *my_level(d)* and *mycousin_level(d)* as follows:

$$my\_level(d) = 1 + \lfloor d \rfloor \quad (1)$$

$$mycousin\_level(d) = my\_level(d) + R \quad (2)$$

where

$$R = 2 \times (d - \lfloor d \rfloor) \quad (3)$$

Let $m = my\_level(d)$ and $n = mycousin\_level(d)$. Intuitively, given a node u and the distance value d, beginning with u, we can go m levels up to reach an ancestor w of u. Then, from w, we can go n levels down to reach a descendant v of w. Referring to the cousin distance definition in Figure 2, $c\_dist(u,v)$ must be equal to the distance value d. Furthermore, all the siblings of u must also be cousins of the siblings of v with the same distance value d. These nodes are identified by their unique identification numbers. To obtain cousin pair items having the form $(L(u), L(v), c\_dist(u,v), occur(u,v))$, we check the node labels of u, v and add up the occurrence numbers for cousin pairs whose corresponding node labels are the same and whose cousin distances are the same. Figure 3 summarizes the algorithm.

Notice that within the for-loop (Step 3 - Step 10) of the algorithm in Figure 3, we find cousin pairs with cousin distance d where d is incremented from 0 to $maxdist$. In Step 8 where a cousin pair with the current distance value d is formed, we check, through node identification numbers, to make sure this cousin pair is not identical to any cousin pair with less distance found in a previous iteration in the loop. This guarantees that only cousin pairs with exact distance d are formed in the current iteration in the loop.

Lemma 1. Algorithm Single_Tree_Mining correctly finds all cousin pair items of T where the cousin pairs have a distance less than or equal to $maxdist$ and an occurrence number greater than or equal to $minoccur$.

Proof. The correctness of the algorithm follows directly from two observations: (i) every cousin pair with distance d where $0 \le d \le maxdist$ is found by the algorithm; (ii) because of Step 9 that eliminates duplicate cousin pairs from consideration, no cousin pair with the same identification numbers is counted twice.

Lemma 2. The time complexity of algorithm Single_Tree_Mining is $O(|T|^2)$.

Procedure: Single_Tree_Mining
Input: A tree T and a maximum distance value allowed, $maxdist$, and a minimum occurrence number allowed, $minoccur$.
Output: All cousin pair items of T where the cousin pairs have a distance less than or equal to $maxdist$ and an occurrence number greater than or equal to $minoccur$.

1. **for** each node p where $children\_set(p) \neq \emptyset$ **do**
2. **begin**
3. **for** each valid distance value $d \leq maxdist$ **do**
4. **begin**
5. let u be a node in $children\_set(p)$;
6. calculate $m = my\_level(d)$ and $n = mycousin\_level(d)$ as defined in Eq. (1), (2);
7. beginning with u, go m levels up to reach an ancestor w and then from w, go n levels down to reach a descendant v of w;
8. combine all siblings of u and all siblings of v to form cousin pairs with the distance value d;
9. if a specific pair of nodes with the distance d has been found previously, don't double-count them;
10. **end**;
11. **end**;
12. add up the occurrence numbers of cousin pairs whose corresponding node labels are the same and whose cousin distances are the same to get qualified cousin pair items of T.

Figure 3. Algorithm for finding frequent cousin pair items in a single tree.

Proof. The algorithm visits each children set of T. For each visited node, it takes at most $O(|T|)$ time to go up and down to locate its cousins. Thus, the time spent in finding all cousin pairs identified by their unique identification numbers is $O(|T|^2)$. There are at most $O(|T|^2)$ such cousin pairs. Through the table lookup, we get their node labels and add up the occurrence numbers of cousin pairs whose distances and corresponding node labels are the same in $O(|T|^2)$ time.

To find all frequent cousin pairs in a set of trees $\{T_1, \ldots, T_k\}$ whose distance is at most *maxdist* and whose support is at least *minsup* for a user-specified *minsup* value, we first find all cousin pair items in each of the trees that satisfy the distance requirement. Then we locate all frequent cousin pairs by counting the number of trees in which a qualified cousin pair item occurs. This procedure will be referred to as Multiple_Tree_Mining and its time complexity is clearly $O(kn^2)$ where $n = \max\{|T_1|, \ldots, |T_k|\}$.

4 Experiments and Results

We conducted a series of experiments to evaluate the performance of the proposed tree mining algorithms, on both synthetic data and phylogenies, run under the Solaris operating system on a SUN Ultra 60 workstation. The tree mining algorithms were implemented using the K programming language (`www.kx.com`). The synthetic data was produced by a C++ program based on the algorithm developed in [19]. This program is able to generate a large number of random trees from the whole tree space. The phylogenies were obtained from TreeBASE, available at `www.treebase.org` [34].

Table 2 summarizes the parameters of our algorithms and their default values used in the experiments. The value of 2 was used for minimum support because the phylogenies in TreeBASE differ substantially and using this support value allowed us to find interesting patterns in the trees. Table 3 lists the parameters and their default values related to the synthetic trees. The $fanout$ of a tree is the number of children of each node in the tree. The $alphabet\_size$ is the total number of distinct node labels these synthetic trees have.

Figure 4 shows how changing the $fanout$ of synthetic trees affects the running time of the algorithm Single_Tree_Mining. 1,000 trees were tested and the average was plotted. The other parameter values are as shown in Table 2 and Table 3. Given a fixed $tree\_size$ value, a large fanout value will result in a small number of children sets, which will consequently reduce the times of executing the outer for-loop of the algorithm, c.f. Step 1 in Figure 3. Therefore, one may expect that the running time of Single_Tree_Mining drops as $fanout$ increases. To our surprise, however, Figure 4 shows that the running time of Sin-

Name	Meaning	Value
$minoccur$	minimum occurrence number of an interesting cousin pair in a tree	1
$maxdist$	maximum distance allowed for an interesting cousin pair	1.5
$minsup$	minimum number of trees in the database that contain an interesting cousin pair	2

Table 2. Parameters and their default values used in the algorithms.

Name	Meaning	Value
$tree\_size$	number of nodes in a tree	200
$database\_size$	number of trees in the database	1,000
$fanout$	number of children of each node in a tree	5
$alphabet\_size$	size of the node label alphabet	200

Table 3. Parameters and their default values related to synthetic trees.

gle_Tree_Mining increases as a tree becomes bushy, i.e. its $fanout$ becomes large. This happens probably because for bushy trees, each node has many siblings and hence more qualified cousin pairs could be generated, cf. Step 8 in Figure 3. As a result, it takes more time in the post-processing stage to aggregate those cousin pairs, cf. Step 12 in Figure 3.

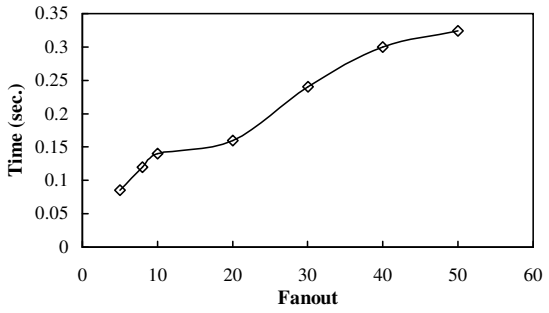

Figure 4. Effect of $fanout$.

Figure 5 shows the running times of Single_Tree_Mining with different $maxdist$ values for

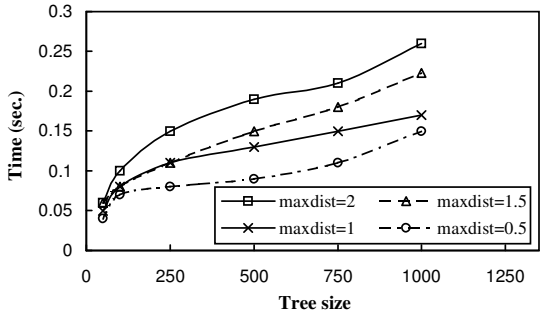

Figure 5. Effect of $maxdist$ and $tree\_size$.

varying node numbers of trees. 1,000 synthetic trees were tested and the average was plotted. The other parameter values are as shown in Table 2 and Table 3. It can be seen from the figure that as $maxdist$ increases, the running time becomes large, because more time will be spent in the inner for-loop of the algorithm for generating cousin pairs, cf. Steps 3 - 10 in Figure 3. We also observed that a lot of time needs to be spent in aggregating qualified cousin pairs in the post-processing stage of the algorithm, cf. Step 12 in Figure 3. This extra time, though not explicitly described by the asymptotic time complexity $O(|T|^2)$ in Lemma 2, is reflected by the graphs in Figure 5.

Figures 6 and 7 show the running times of Multiple_Tree_Mining when applied to 1 million synthetic trees and 1,500 phylogenies obtained from TreeBASE, respectively. Each phylogeny has between 50 and 200 nodes and each node has between 2 and 9 children (most internal nodes have 2 children). The size of the node label alphabet is 18,870. The other parameter values are as shown in Table 2 and Table 3. We see from Figure 7 that Multiple_Tree_Mining can find all frequent cousin pair items in the 1,500 phylogenetic trees in less than 150 seconds. The algorithm scales up well—its running time increases linearly with increasing number of trees (Figure 6).

5 Applications

In this section we describe several applications of our techniques, showing how they can be used to (i) discover co-occurring patterns in multiple phylogenies; (ii) evaluate the quality of a consensus tree obtained from equally parsimonious trees with the same taxa (leaf nodes); and (iii) find kernel trees from groups of phylogenies with different taxa.

5.1 Discovering Co-occurring Patterns in Multiple Phylogenies

We applied Multiple_Tree_Mining to the phylogenies associated with each study in TreeBASE to discover co-

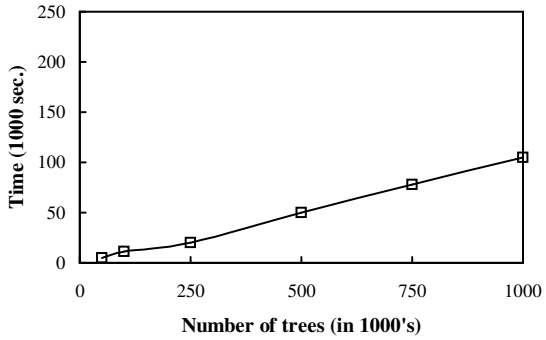

Figure 6. Effect of $database\_size$ for synthetic trees.

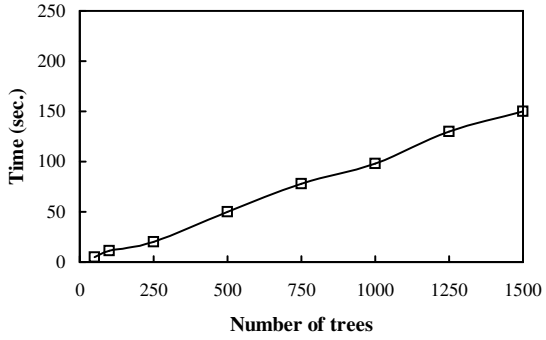

Figure 7. Effect of $database\_size$ for phylogenies.

occurring patterns in these phylogenies. The parameter values used in the algorithm are as shown in Table 2. Figure 8 shows some results for the phylogenies reported in the study [11] maintained in TreeBASE. These phylogenies were constructed for 8 seed plants (taxa): Cycadales, Ginkgoales, Coniferales, Ephedra, Welwitschia, Gnetum, Angiosperms and Outgroup_to_Seed_Plants.

There are several interesting cousin pairs in Figure 8. For example, (Ginkgoales, Ephedra) is a frequent cousin pair with distance 1.5, which is highlighted by an underscore and occurs in the two trees in the two right windows in the figure. (Gnetum, Welwitschia) is another frequent cousin pair with distance 0, which is highlighted by a bullet and occurs in all four trees in the figure. These frequent cousin pairs show evolutionary associations between the taxa studied in [11].

5.2 Evaluating the Quality of Consensus Trees

One important subject in phylogeny is to automatically infer or reconstruct phylogenetic trees from a set of molecular sequences or species. The most commonly used method is based on the maximum parsimony principle [14]. This method often generates multiple trees rather than a single tree for the input sequences or species. When the number of equally parsimonious trees is too large to suggest an informative evolution hypothesis, a consensus tree is sought to summarize the set of parsimonious trees. Sometimes the set is divided into several clusters and a consensus tree for each cluster is derived [37].

There are five most popular methods for generating consensus trees: Adams [1], strict [9], majority [26], semi-strict [5], and Nelson [30]. Here, we evaluate the quality of the consensus tree generated by each of the above five methods. The quality is measured by considering the cousin pairs shared between the consensus tree and the original parsimonious trees from which the consensus tree is generated.

Specifically, let C be a consensus tree and let T be an original parsimonious tree. We define the similarity score between C and T, denoted $\delta(C,T)$, as

$$\delta(C,T) = \sum_{i=1}^{k} \frac{1}{1 + |c\_dist_C(cp_i) - c\_dist_T(cp_i)|} \quad (4)$$

where $|.|$ is the absolute value of the indicated number. Each cp_i, $1 \leq i \leq k$, is a cousin pair whose node labels occur in both C and T; $|c\_dist_C(cp_i) - c\_dist_T(cp_i)|$ is the difference of the cousin distances of the cp_i shared by C and T. Thus, if the shared cousin pair cp_i has the same distance in C and T, it will contribute 1 to $\delta(C,T)$. If its distance is different in C and T, the value it contributes to $\delta(C,T)$ will be less than 1.

Let S be the set of original parsimonious trees from which the consensus tree C is generated. The average similarity score of the consensus tree C with respect to the set S is

$$\Delta(C,S) = \frac{\sum_{T \in S} \delta(C,T)}{|S|} \quad (5)$$

where $|S|$ is the total number of trees in the set S. The higher the average similarity score C has, the better consensus tree C is.

Figure 9 compares average similarity scores of the consensus trees generated by the five methods mentioned above for varying number of equally parsimonious trees. The parameter values used by our algorithm for finding the cousin pairs are as shown in Table 2. The parsimonious trees were generated by the PHYLIP tool [13] using the first 500 nucleotides extracted from six genes representing paternally, maternally, and biparentally inherited regions of the genome among 16 species of Mus [24]. It can be seen from Figure 9 that the majority consensus method is the best, yielding consensus trees with the highest scores.

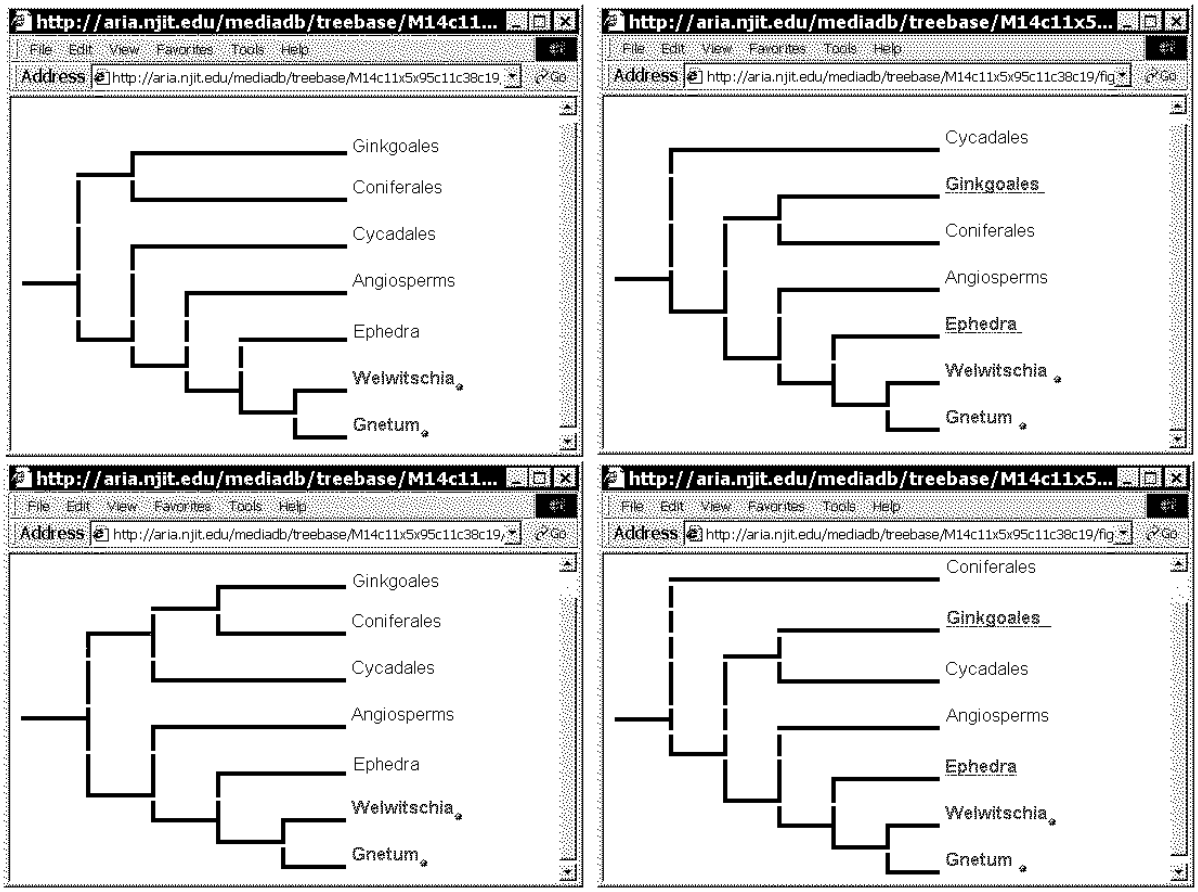

Figure 8. Discovering co-occurring patterns in multiple phylogenies.

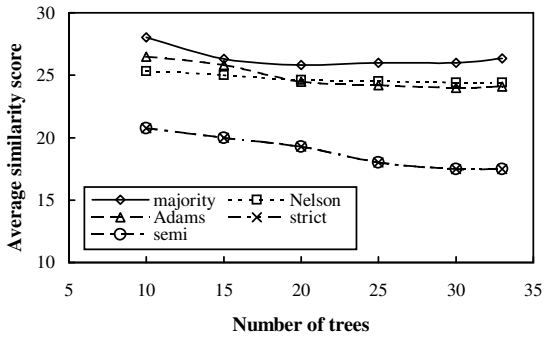

Figure 9. Comparing the quality of consensus trees using frequent cousin pairs.

5.3 Finding Kernel Trees from Groups of Phylogenies

Existing phylogenetic distance measures, such as those implemented in the widely used COMPONENT tool [31], are designed for comparing evolutionary trees with the same taxa (leaf nodes). However, some applications in phylogeny, such as supertree construction [38], are concerned with assembling information from smaller phylogenies that share some but not necessarily all taxa in common. The COMPONENT tool doesn't work for these phylogenies.

We propose here a distance measure for comparing phylogenetic trees based on the frequent cousin pairs found in the trees. Specifically, let T_1 and T_2 be two trees. Let $cpi(T_1)$ contain all the cousin pair items of T_1 and let $cpi(T_2)$ contain all the cousin pair items generated from T_2. We define the *tree distance* of T_1 and T_2, denoted $t\_dist(T_1, T_2)$, as

$$t\_dist(T_1, T_2) = \frac{|cpi(T_1) \cap cpi(T_2)|}{|cpi(T_1) \cup cpi(T_2)|} \quad (6)$$

Depending on whether the cousin distance and the number of occurrences of a cousin pair in a tree are considered, we have four different types of cousin pair items in the tree. Consequently we obtain four different tree distance measures. We represent them by $t\_dist_{null}(T_1, T_2)$ (considering neither the cousin distance nor the occurrence number in each tree), $t\_dist_{cdist}(T_1, T_2)$ (considering the cousin distance only in each tree), $t\_dist_{occ}(T_1, T_2)$ (considering the occurrence number only in each tree), and $t\_dist_{occ\_cdist}(T_1, T_2)$ (considering both the cousin distance and the occurrence number in each tree), respectively.

For example, referring to the trees T_2 and T_3 in Figure 1, we have $t\_dist_{null}(T_2, T_3) = \frac{4}{12} = 0.33$, $t\_dist_{cdist}(T_2, T_3) = \frac{2}{16} = 0.125$, $t\_dist_{occ}(T_2, T_3) = \frac{4}{12} = 0.33$, $t\_dist_{occ\_cdist}(T_2, T_3) = \frac{2}{16} = 0.125$.[2]

Using the proposed tree distance measure, we can find kernel trees from groups of phylogenies. Specifically, we consider k, $2 \leq k \leq 5$, groups of phylogenies, referred to as $jset_1, \ldots, jset_k$, where the phylogenies were generated by PHYLIP [13] using the LSU rDNA sequences representing 32 ascomycetes [23]. Data in the same group are parsimonious trees for the same taxa while different groups share some but not all taxa in common. We find the kernel trees $best_1, \ldots, best_k$ such that the average pairwise distance between the kernel trees is minimized and $best_i$ comes from $jset_i$. The distance measure used is the tree distance $t\_dist_{occ\_cdist}$ described above and the parameter values are as shown in Table 2. Figure 10 shows the time spent in finding the kernel trees as a function of the group number k. The found kernel trees could constitute a good starting point in building a supertree for the phylogenies in the k groups.

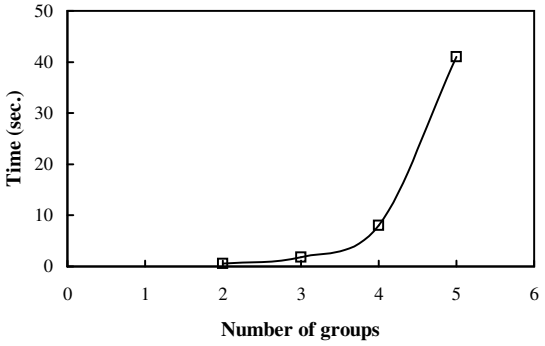

Figure 10. Time spent in finding kernel trees.

[2]The intersection and union of two sets of cousin pair items take into account the occurrence numbers in them. For example, suppose $cpi(T_1) = \{(a, b, m, occur1)\}$ and $cpi(T_2) = \{(a, b, m, occur2)\}$. Then $cpi(T_1) \cap cpi(T_2) = \{(a, b, m, \min(occur1, occur2))\}$ and $cpi(T_1) \cup cpi(T_2) = \{(a, b, m, \max(occur1, occur2))\}$.

6 Extensions to Graphs

Some phylogenetic tree reconstruction methods such as MP [14] and ML [12] may produce unrooted unordered labeled trees. These trees are also known as free trees or undirected acyclic graphs (UAGs) [41, 49]. In this section we discuss an extension of our single tree mining algorithm to find frequent cousin pairs in one such graph.

In UAGs, the cousin distance between two nodes u, v is modified as follows:

$$c\_dist(u, v) = \frac{n}{2} - 1 \quad (7)$$

where n is the number of edges between u and v. Thus, given a cousin distance value d, the number of edges, n, between u and v can be calculated as follows:

$$n = (d + 1) \times 2 \quad (8)$$

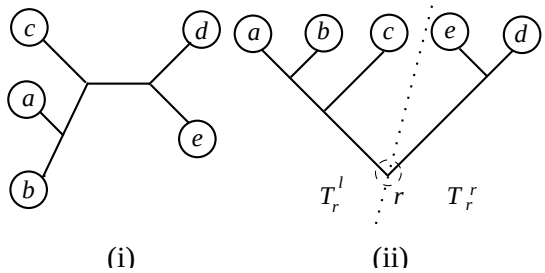

Figure 11. Illustration of converting an undirected acyclic graph (shown in (i)) to a rooted tree with node r being the root (shown in (ii)).

Given an undirected acyclic graph G, together with the maximum distance allowed ($maxdist$) and the minimum occurrence number allowed ($minoccur$), we find frequent cousin pairs in G by arbitrarily choosing an edge e in G and putting an artificially created node r on e so that G becomes a rooted tree T_r with r being the root (see Figure 11). This tree consists of two subtrees, one being on the left side of r, denoted T_r^l, and the other being on the right side of r, denoted T_r^r. We then modify the Single_Tree_Mining algorithm in Figure 3 and apply it to T_r as follows.

Consider each valid distance value $d \leq maxdist$. Let u be a node in a children set of T_r. We go m levels up to reach an ancestor w, and then from w, we go n levels down to get a cousin v of u where

$$m + n = (d + 1) \times 2 \quad (9)$$

and $1 \leq m, n \leq [(d + 1) \times 2] - 1$. Referring to Eq. (8), the $[(d + 1) \times 2]$ represents the number of edges between u

and v. Thus, instead of only considering $my\_level(d)$ and $mycousin\_level(d)$ as defined in Eq. (1), (2), we consider here all combinations of m, n satisfying Eq. (9). For example, suppose d is 2. Possible combinations of (m, n) include (1, 5), (2, 4), (3, 3), (4, 2), and (5, 1). The above calculation is correct when both u and v are in T_r^l, or both u and v are in T_r^r. Otherwise we have to consider the additional edge created due to the insertion of the root r. Specifically, suppose u is in T_r^l and v is in T_r^r. Then the m, n used in traversing the tree should be modified as follows:

$$m + n = [(d + 1) \times 2] + 1 \tag{10}$$

and $1 \leq m$, $n \leq [(d + 1) \times 2]$, to take into account the additional edge inserted in T_r. Clearly the time complexity of this algorithm is $O(|G|^2)$. One can easily extend this algorithm to find frequent cousin pairs in multiple graphs.

7 Conclusion and Future Work

We presented new algorithms for finding and extracting frequent cousin pairs with varying distances from a single tree or multiple trees. The software for these algorithms can be downloaded from `http://cs.nyu.edu/cs/faculty/shasha/papers/cousins.k`. Our algorithm for the single tree mining method, described in Section 3, is a quadratic-time algorithm. We suspect the best-case time complexity for finding all frequent cousin pairs in a tree is also quadratic. We plan to investigate alternative approaches (e.g. dynamic programming) to find these patterns in phylogenetic trees.

Notice that our approach differs from the work on computing least common ancestors of two nodes in a tree (e.g. [4, 17]) in that we use the definition of cousin distance to guide the search and mining process. Specifically given a cousin distance value d, beginning with a node u, we move up to find an ancestor w of u, and then from w we move down to reach a cousin v of u. The number of steps to move up and down is completely determined by the given distance value d. Thus, we systematically enumerate the cousins rather than taking random pairs of nodes and finding out what kind of cousins they are.

We also introduced a similarity measure based on frequent cousin pairs and used it to compare five popular methods for consensus tree generation. As far as we know, this is the first attempt to evaluate the quality of consensus trees through a quantitative measure. Other possible measures could be based on the various distances for phylogenetic trees as described in [31]. We plan to compare our approach with these other methods. Other future work includes (i) extending the proposed techniques to trees whose edges have weights, and (ii) finding different types of patterns in the trees and using them in phylogenetic data clustering as well as other applications (e.g. XML query processing and the analysis of metabolic pathways [18]).

Acknowledgment

We thank William Piel for useful discussions on TreeBASE. We also thank the ICDE anonymous reviewers for their thoughtful comments, which helped to improve the quality and presentation of this paper.

References

[1] E. N. Adams. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21:390–397, 1972.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.

[3] T. Asai, H. Arimura, K. Abe, S. Kawasoe, and S. Arikawa. Online algorithms for mining semi-structured data streams. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 27–34, 2002.

[4] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Theoretical Informatics Symposium*, pages 88–94, 2000.

[5] K. Bremer. Combinable component consensus. *Cladistics*, 6:369–372, 1990.

[6] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. Counting twig matches in a tree. In *Proceedings of the 17th IEEE International Conference on Data Engineering*, pages 595–604, 2001.

[7] R. Cole, R. Hariharan, and P. Indyk. Tree pattern matching and subset matching in deterministic $o(nlog^3m)$ time. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 245–254, 1999.

[8] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artifi cial Intelligence Research*, 1:231–255, 1994.

[9] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classifi cation*, 1:7–28, 1985.

[10] L. Dehaspe, H. T. T. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[11] J. A. Doyle and M. J. Donoghue. Fossils and seed plant phylogeny reanalyzed. *Brittonia*, 33:89–106, 1992.

[12] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[13] J. Felsenstein. PHYLIP – Phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

[14] W. Fitch. Toward the defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 1971.

[15] M. Garofalakis and A. Kumar. Correlating XML data streams using tree-edit distance embeddings. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2003.

[16] Genealogy.com. `http://www.genealogy.com/16_cousn.html`.

[17] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[18] M. Heymans and A. K. Singh. Deriving phylogenetic trees from the similarity analysis of metabolic pathways. In *Proceedings of the 11th International Conference on Intelligent Systems for Molecular Biology*, pages 138–146, 2003.

[19] S. Holmes and P. Diaconis. Random walks on trees and matchings. *Electronic Journal of Probability*, 7, 2002.

[20] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.

[21] P. Kilpelainen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.

[22] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 1st IEEE International Conference on Data Mining*, pages 313–320, 2001.

[23] H. T. Lumbsch, R. Lindemuth, and I. Schmitt. Evolution of filamentous ascomycetes inferred from LSU rDNA sequence data. *Plant Biology*, 2:525–529, 2000.

[24] B. L. Lundrigan, S. Jansa, and P. K. Tucker. Phylogenetic relationships in the genus mus, based on paternally, maternally, and biparentally inherited characters. *Systematic Biology*, 51:23–53, 2002.

[25] N. Mamoulis. Mining frequent patterns in XML documents. Unpublished manuscript, 2002.

[26] T. Margush and F. R. McMorris. Consensus n-trees. *Bull. Math. Biol.*, 43:239–244, 1981.

[27] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering*, 2002.

[28] C. Moh, E. Lim, and W. Ng. DTD-miner: A tool for mining DTD from XML documents. In *Proceedings of the 2nd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2000.

[29] R. Nayak, R. Witt, and A. Tonev. Data mining and XML documents. In *Proceedings of the International Conference on Internet Computing*, 2002.

[30] G. Nelson. Cladistic analysis and synthesis: Principles and definitions, with a historical note on Adanson's Famille des Plantes (1763-1764). *Systematic Zoology*, 28:1–21, 1979.

[31] R. D. M. Page. COMPONENT user's manual (release 1.5), 1989. University of Auckland, Auckland.

[32] R. D. M. Page and E. C. Holmes. *Molecular Evolution: A Phylogenetic Approach*. Blackwell Science, 1998.

[33] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 81–90, 2002.

[34] M. J. Sanderson, M. J. Donoghue, W. H. Piel, and T. Erikson. Treebase: A prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *American Journal of Botany*, 81(6):183, 1994.

[35] R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.

[36] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 39–52, 2002.

[37] C. Stockham, L. Wang, and T. Warnow. Statistically based postprocessing of phylogenetic analysis by clustering. In *Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology*, pages 285–293, 2002.

[38] Supertree server. `http://genome.cs.iastate.edu/supertree/`.

[39] J. T. L. Wang, H. Shan, D. Shasha, and W. H. Piel. TreeRank: A similarity measure for nearest neighbor searching in phylogenetic databases. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, 2003.

[40] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and C. Y. Chang. Automated discovery of active motifs in multiple RNA secondary structures. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 70–75, 1996.

[41] J. T. L. Wang, K. Zhang, G. Chang, and D. Shasha. Finding approximate patterns in undirected acyclic graphs. *Pattern Recognition*, 35(2):473–483, 2002.

[42] K. Wang and H. Liu. Discovering typical structures of documents: A road map approach. In *Proceedings of the 21st ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 146–154, 1998.

[43] K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):353–371, 2000.

[44] Q. Y. Wang, J. X. Yu, and K. Wong. Approximate graph schema extraction for semi-structured data. In *Proceedings of the 7th International Conference on Extending Database Technology*, pages 302–316, 2000.

[45] X. Wang, J. T. L. Wang, D. Shasha, B. A. Shapiro, I. Rigoutsos, and K. Zhang. Finding patterns in three-dimensional graphs: Algorithms and applications to scientific data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):731–749, 2002.

[46] X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[47] K. Yoshida and H. Motoda. Clip: Concept learning from inference patterns. *Artifi cial Intelligence*, 75(1):63–92, 1995.

[48] M. J. Zaki. Effi ciently mining frequent trees in a forest. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2002.

[49] K. Zhang, J. T. L. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs. *International Journal of Foundations of Computer Science*, 7(1):43–57, 1996.