An Even Faster and More Unifying Algorithm for Comparing Trees via Unbalanced Bipartite Matchings*

Ming-Yang Kao† Tak-Wah Lam‡ Wing-Kin Sung§ Hing-Fung Ting‡

February 1, 2008

Abstract

A widely used method for determining the similarity of two labeled trees is to compute a maximum agreement subtree of the two trees. Previous work on this similarity measure is only concerned with the comparison of labeled trees of two special kinds, namely, *uniformly labeled* trees (i.e., trees with all their nodes labeled by the same symbol) and *evolutionary* trees (i.e., leaf-labeled trees with distinct symbols for distinct leaves). This paper presents an algorithm for comparing trees that are labeled in an arbitrary manner. In addition to this generality, this algorithm is faster than the previous algorithms.

Another contribution of this paper is on maximum weight bipartite matchings. We show how to speed up the best known matching algorithms when the input graphs are node-unbalanced or weight-unbalanced. Based on these enhancements, we obtain an efficient algorithm for a new matching problem called the *hierarchical bipartite matching* problem, which is at the core of our maximum agreement subtree algorithm.

1 Introduction

A *labeled* tree is a rooted tree with an arbitrary subset of nodes labeled with symbols. In recent years, many algorithms for comparing such trees have been developed for diverse application areas including biology [8, 19, 23], chemistry [25], linguistics [9, 21], computer vision [18], and structured text databases [16, 17, 20].

A widely used measure of the similarity of two labeled trees is the notion of a maximum agreement subtree defined as follows. A labeled tree R is a *label-preserving homeomorphic* subtree of another labeled tree T if there exists a one-to-one mapping f from the nodes of R to those of T such that for any nodes u, v, w of R, (1) u and $f(u)$ have the same label; and (2) w is the least

*This paper combines results of three conference papers: (1) A faster and unifying algorithm for comparing trees. In *Proceedings of the 11th Symposium on Combinatorial Pattern Matching*, pages 129-142, 2000. (2) Unbalanced and hierarchical bipartite matchings with applications to labeled tree comparison. In *Proceedings of the 11th International Symposium on Algorithms and Computation,* pages 479-490, 2000. (3) A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees. In *Proceedings of the 8th Annual European Symposium on Algorithms,* pages 438-449, 1999.

†Department of Computer Science, Yale University, New Haven, CT 06520, USA; kao-ming-yang@cs.yale.edu. Research supported in part by NSF Grant CCR-9531028.

‡Department of Computer Science and Information Systems, University of Hong Kong, Hong Kong; {twlam, hfting}@cs.hku.hk. Research supported in part by Hong Kong RGC Grant HKU-7027/98E.

§E-Business Technology Institute, University of Hong Kong, Hong Kong; wksung@eti.hku.hk.

common ancestor of u and v if and only if $f(w)$ is the least common ancestor of $f(u)$ and $f(v)$. Let T_1 and T_2 be two labeled trees. An *agreement* subtree of T_1 and T_2 is a labeled tree which is also a label-preserving homeomorphic subtree of the two trees. A *maximum* agreement subtree is one which maximizes the number of labeled nodes. Let $\text{MAST}(T_1, T_2)$ denote the number of labeled nodes in a maximum agreement subtree of T_1 and T_2.

In the literature, many algorithms for computing a maximum agreement subtree have been developed. These algorithms focus on the special cases where T_1 and T_2 are either (1) *uniformly labeled* trees, i.e., trees with all their nodes unlabeled, or equivalently, labeled with the same symbol or (2) *evolutionary trees* [13], i.e., leaf-labeled trees with distinct symbols for distinct leaves.

We denote n as the number of nodes in the labeled trees T_1 and T_2, and d as the maximum degree of T_1 and T_2. For uniformly labeled trees T_1 and T_2, Chung [3] gave an algorithm to determine whether T_1 is a label-preserving homeomorphic subtree of T_2 using $O(n^{2.5})$ time. Gupta and Nishimura [11] gave an algorithm which actually computes a maximum agreement subtree of T_1 and T_2 in $O(n^{2.5} \log n)$ time. For evolutionary trees, Steel and Warnow [24] gave the first polynomial-time algorithm for computing a maximum agreement subtree. Farach and Thorup [7] improved the time complexity from $O(n^{4.5} \log n)$ to $O(n^{1.5} \log n)$. Faster algorithms for the case $d = O(1)$ were also discovered. The algorithm of Farach, Przytycka and Thorup [6] runs in $O(\sqrt{d} n \log^3 n)$ time, and that of Kao [14] takes $O(nd^2 \log^2 n \log d)$ time. Cole *et al.* [4] gave an $O(n \log n)$-time algorithm for the case where T_1 and T_2 are binary trees. Przytycka [22] attempted to generalize the algorithm of Cole *et al.* so that the degree-2 restriction could be removed with the running time being $O(\sqrt{d} n \log n)$.

For *unrestricted* labeled trees (i.e., trees where labels are not restricted to leaves and may not be distinct), little work has been reported, but they have applications in several contexts [1, 18]. For example, labeled trees are used to represent sentences in a structural text database [16, 17, 20] and querying such a database involves comparison of trees; an XML document can also be represented by a labeled tree [1]. Instead of solving special cases, this paper gives an algorithm to compute $\text{MAST}(T_1, T_2)$ where T_1 and T_2 are unrestricted labeled trees. As detailed below, our algorithm not only is more general but also uniformly improves or matches the previously best algorithms for subtree homeomorphism and evolutionary tree comparison.

Let Δ_{T_1,T_2} (or simply Δ when the context is clear) $= \sum_{u \in T_1} \sum_{v \in T_2} \delta(u, v)$ where $\delta(u, v) = 1$ if nodes u and v are labeled with the same symbol, and 0 otherwise. Our algorithm computes $\text{MAST}(T_1, T_2)$ in $O(\sqrt{d} \Delta \log \frac{2n}{d})$ time. Thus, if T_1 and T_2 are uniformly labeled trees, then $\Delta \leq n^2$ and the time complexity of our algorithm is $O(\sqrt{d} n^2 \log \frac{2n}{d})$, which is faster than the Gupta-Nishimura algorithm [11] for any d. If T_1 and T_2 are evolutionary trees, then $\Delta \leq n$ and the time complexity of our algorithm is $O(\sqrt{d} n \log \frac{2n}{d})$, which is better than the $O(\sqrt{d} n \log n)$ bound claimed by Przytycka [22]. In particular, our algorithm can attain the $O(n \log n)$ bound for binary trees [4]. Also for general evolutionary trees, our algorithm runs in $O(n^{1.5})$ time since $\sqrt{d} n \log \frac{2n}{d} = O(n^{1.5})$ for any degree d. This is faster than the $O(n^{1.5} \log n)$ time of the Farach-Thorup algorithm [7].

The efficiency achieved by our MAST algorithm is based on improved algorithms for computing maximum weight matchings of bipartite graphs that satisfy some structural properties. Let $G = (X, Y, E)$ be a bipartite graph with positive integer weights on its edges. Denote by n, m, N, and W the number of nodes, the number of edges, the maximum edge weight, and the total edge weight of G, respectively. The best known algorithm for computing maximum weight bipartite matchings was given by Gabow and Tarjan [10], which takes $O(\sqrt{n} m \log nN)$ time. For some applications where the total edge weight is small (say, $W = O(m)$), Kao *et al.* [15] gave a slightly faster algorithm that runs in $O(\sqrt{n} W)$ time. Intuitively, a bipartite graph is *node-unbalanced* if

there are much fewer nodes on one side than the other. It is *weight-unbalanced* if its total weight is dominated by the edges incident to a few nodes; we call these nodes the *dominating* nodes. In this paper, we show how to enhance these two matching algorithms when the input graphs are either node-unbalanced or weight-unbalanced.

The node-unbalanced property has many practical applications (see, e.g., [12]) and has been exploited to improve various graph algorithms. For example, Ahuja *et al.* [2] adapted several bipartite network flow algorithms such that the running times depend on the number of nodes in the smaller side of the input bipartite graph instead of the total number of nodes. Tokuyama and Nakano used this property to reduce the time complexity of the minimum cost assignment problem [27] and the Hitchcock transportation problem [26]. This paper presents similar improvements for maximum weight matching. Specifically, we show that the running time of the matching algorithms of Gabow and Tarjan [10] and Kao *et al.* [15] can be improved to $O(\min\{\sqrt{n_s}m \log n_s N, m + n_s^{2.5} \log n_s N\})$ and $O(\sqrt{n_s}W)$, respectively, where n_s is the number of nodes in the smaller side of the input bipartite graph.

The weight-unbalanced property is exploited in another way. Given a weight-unbalanced bipartite graph G, let G' be the subgraph of G with its dominating nodes removed. Note that G' has a total weight much smaller than G does. Based on the $O(\sqrt{n}W)$-time matching algorithm of Kao *et al.* [15], finding the maximum weight matching of G' is much faster than finding one of G. To take advantage of this fact, we design an efficient algorithm that finds a maximum weight matching of G from that of G'. This algorithm is substantially faster than applying directly the $O(\sqrt{n}W)$-time matching algorithm on G.

These results for unbalanced graphs provide a basis for solving a new matching problem called the *hierarchical bipartite matching* problem. This matching problem is at the core of our MAST algorithm and is defined as follows. Let T be a rooted tree. Denote r as the root of T. Let $C(u)$ denote the set of children of node u. Every node u of T is associated with a positive integer $w(u)$ and a weighted bipartite graph G_u satisfying the following properties:

- $w(u) \geq \sum_{v \in C(u)} w(v)$.
- $G_u = (X_u, Y_u, E_u)$ where $X_u = C(u)$. Each edge of G_u has a positive integer weight, and there is no isolated node. For any node $v \in X_u$, the total weight of all the edges incident to v is at most $w(v)$. Thus, the total weight of the edges in G_u is at most $w(u)$.

See Figure 1 for an example. For any weighted bipartite graph G, let $\text{MWM}(G)$ denote a maximum weight matching of G. The *hierarchical matching problem* is to compute $\text{MWM}(G_u)$ for all internal nodes u of T. Let $b = \max_{u \in T}\{\min\{|X_u|, |Y_u|\}\}$ and $e = \sum_{u \in T} |E_u|$. The problem can be solved by applying directly our results for node-unbalanced graphs; for example, it can be solved in $O(\sum_{u \in T} \sqrt{b}|E_u| \log bw(u)) = O(\sqrt{b}e \log w(r))$ time using the enhanced Gabow-Tarjan algorithm. However, this time complexity is not yet satisfactory. When comparing labeled trees, we often encounter instances of the hierarchical bipartite matching problem with e being very large; in particular, e is asymptotically much greater than $w(r)$. We further improve the running time to $O(\sqrt{b}w(r) + e)$ by making additional use of our technique for weight-unbalanced graphs and exploiting trade-offs between the size of the bipartite graphs involved and their total edge weight.

The rest of the paper is organized as follows. Section 2 details our techniques of speeding up the existing algorithms for unbalanced graphs. Section 3 gives an efficient algorithm for solving the hierarchical matching problem. Finally, Section 4 describes our algorithm for computing maximum agreement subtrees.

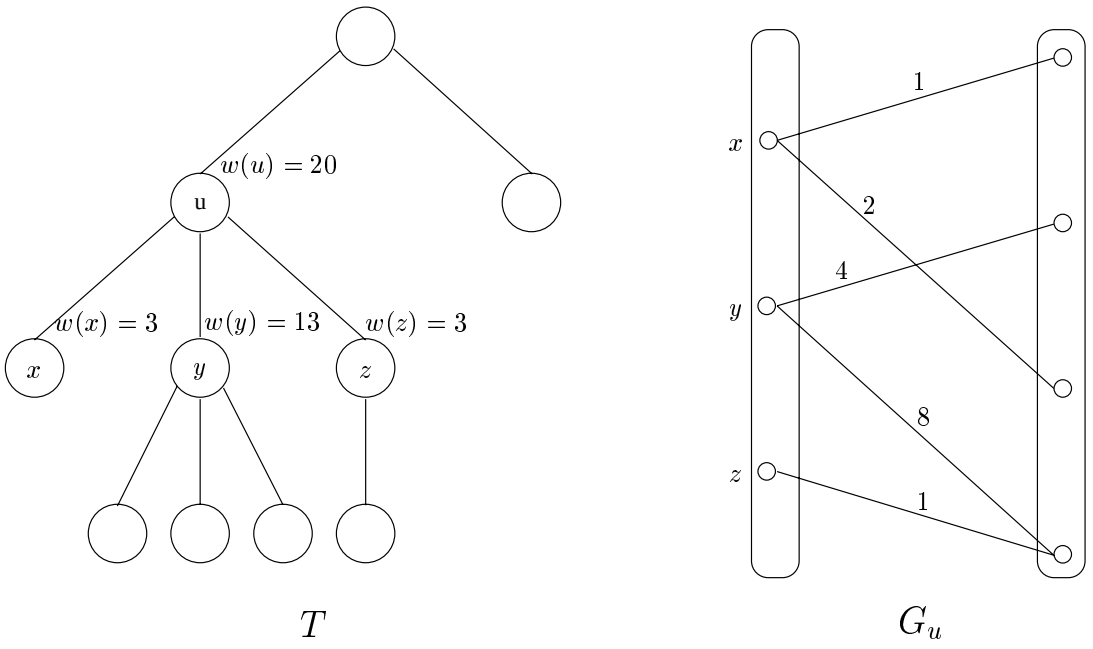

Figure 1: T is an instance of the hierarchical bipartite matching problem. G_u is the bipartite graph associated with the node u.

2 Maximum weight matching of unbalanced graphs

Throughout this section, let $G = (X, Y, E)$ be a weighted bipartite graph with no isolated nodes. Let $n = |X| + |Y|$, $n_s = \min\{|X|, |Y|\}$, $m = |E|$, N be the largest edge weight, and W be the total edge weight.

Suppose every edge of G has a positive integer weight. Gabow and Tarjan [10] and Kao *et al.* [15] gave an $O(\sqrt{n}m\log(nN))$-time algorithm and an $O(\sqrt{n}W)$-time one to compute $\textsc{mwm}(G)$, respectively.

2.1 Matchings of node-unbalanced graphs

The following theorem speeds up the computation of $\textsc{mwm}(G)$ if G is node-unbalanced.

Theorem 1.

1. $\textsc{mwm}(G)$ *can be computed in* $O(\sqrt{n_s}m\log(n_sN))$ *time.*
2. $\textsc{mwm}(G)$ *can be computed in* $O(m + n_s^{2.5}\log(n_sN))$ *time.*
3. $\textsc{mwm}(G)$ *can be computed in* $O(\sqrt{n_s}W)$ *time.*

Proof. Without loss of generality, we assume $n_s = |X| \le |Y|$. The statements are proved as follows.

Statement 1. For any node v in G, let $\alpha(v)$ be the number of edges incident to v. Suppose $Y = \{y_1, y_2, \ldots, y_{kn_s+r}\}$ where $k \ge 1$, $0 \le r < n_s$, and $\alpha(y_1) \le \alpha(y_2) \le \cdots \le \alpha(y_{kn_s+r})$. We partition

Y into $Y_0 = \{y_1, \ldots, y_r\}, Y_1 = \{y_{r+1}, \ldots, y_{r+n_s}\}, \ldots,$ and $Y_k = \{y_{r+(k-1)n_s+1}, \ldots, y_{r+kn_s}\}$. Note that except Y_0, every set has n_s nodes.

For any $Y' \subseteq Y$, denote $G(Y')$ as the subgraph of G induced by all the edges incident to Y'. Suppose that M_i is a maximum weight matching of $G(Y_0 \cup Y_1 \cup \cdots \cup Y_i)$. Let $Y_{M_i} = \{y \mid (x, y) \in M_i\}$. Note that a maximum weight matching of $G(Y_{M_i} \cup Y_{i+1})$ is also one of $G(Y_0 \cup Y_1 \cup \cdots \cup Y_{i+1})$. Therefore, we can compute MWM(G) using the following algorithm:

- Step 1. Compute a maximum weight matching M_0 of $G(Y_0)$.
- Step 2. For $i = 1$ to k,

  let $Y_{M_{i-1}} = \{y \mid (x, y) \in M_{i-1}\}$;
  compute a maximum weight matching M_i of $G(Y_{M_{i-1}} \cup Y_i)$.
- Step 3. Return M_k.

The running time is analyzed below. Let $\alpha(Y')$ be the total number of edges in $G(Y')$. For $1 \leq i \leq k$, $\alpha(Y_{M_{i-1}}) \leq \alpha(Y_i)$, and $\alpha(Y_{M_{i-1}} \cup Y_i) \leq 2\alpha(Y_i)$. Using the matching algorithm by Gabow and Tarjan [10], we can compute MWM($G(Y_{M_{i-1}} \cup Y_i)$) in $O(\sqrt{|Y_{M_{i-1}} \cup Y_i|}\alpha(Y_i) \log(|Y_{M_{i-1}} \cup Y_i|N))$ time. Note that $|Y_{M_{i-1}}| \leq n_s$ and $|Y_i| = n_s$. Hence, the whole algorithm uses $O(\sum_{i=1}^{k} \sqrt{n_s}\alpha(Y_i) \log(n_s N))$ $= O(\sqrt{n_s} m \log(n_s N))$ time.

Statement 2. Since we suppose $|X| \leq |Y|$, any matching of G contains at most $|X| = n_s$ edges. Thus, for every $u \in X$, we can discard the edges incident to u that are not among the n_s heaviest ones; the remaining n_s^2 edges must still contain a maximum weight matching of G. Note that we can find these n_s^2 edges in $O(m)$ time, and from Statement 1, we can compute MWM(G) from them in $O(\sqrt{n_s} n_s^2 \log(n_s N))$ time. The total time taken is $O(m + n_s^{2.5} \log(n_s N))$.

Statement 3. The algorithm in the proof of Statement 1 can be adapted to find MWM(G) in $O(\sqrt{n_s} W)$ time by using the $O(\sqrt{n} W)$-time matching algorithm of Kao *et al.* [15] to compute each M_i. For any $Y' \subseteq Y$, we redefine $\alpha(Y')$ to be the total weight of edges incident to Y. Then we can use the same analysis to show that the adapted algorithm runs in $O(\sum_{i=1}^{k} \sqrt{n_s}\alpha(Y_i)) = O(\sqrt{n_s} W)$ time. □

2.2 Matchings of weight-unbalanced graphs

We show how to speed up the matching algorithm of Kao *et al.* [15] when the input graph G is weight-unbalanced. The key technique is stated in Lemma 2 below. The following example illustrates how this lemma can help. Suppose that G has $O(1)$ dominating nodes. Let G' be the subgraph of G with the dominating nodes removed. Let W' be the total edge weight of G'. Since G is weight-unbalanced, we further assume $W' = o(W)$. To compute MWM(G), we can first use Theorem 1(3) to compute MWM(G') in $O(\sqrt{n_s} W')$ time and then use Lemma 2 to compute MWM(G) from MWM(G') in $O(m \log n_s)$ time. The total running time is $O(m \log n_s + \sqrt{n_s} W') = o(\sqrt{n_s} W)$, which is smaller than the running time of using the algorithm of Kao *et al.* [15] to find MWM(G) directly.

Lemma 2. *Let $H = \{x_1, x_2, \ldots, x_h\}$ be a subset of h nodes of X. Let $G - H$ be the subgraph of G constructed by removing the nodes in H. Denote by E' the set of edges in $G - H$. Given* MWM($G - H$)*, we can compute* MWM(G) *in $O(|E| + (h^2|E'| + h^3) \log n_s)$ time.*

Proof. First, we show that using $O(|E|)$ time, we can find a set Υ of only $O(\min\{h|E'|+h^2, n_s^2\})$ edges such that Υ still contains a maximum weight matching of G. In the proof of Theorem 1(2), it has already been shown that we can find in $O(|E|)$ time a set of $O(n_s^2)$ edges that contains $\textsc{mwm}(G)$. Thus, it suffices to find in $O(|E|)$ time another set of $O(h|E'|+h^2)$ edges that contains $\textsc{mwm}(G)$; Υ is just the smaller of these two sets. Let Y' be the subset of nodes of Y that are endpoints of E'. For any $x_i \in H$, we select, among the edges incident to x_i, a subset of edges E_i, which is the union of the following two sets:

- $\{(x_i, y) \mid y \in Y'\}$;
- $\{(x_i, y) \mid (x_i, y)$ is among the h heaviest edges with $y \notin Y'\}$.

Observe that $E' \cup E_1 \cup \cdots \cup E_h$ must contain a maximum weight matching of G, and these $|E' \cup E_1 \cup \cdots E_h| = O(h|E'|+h^2)$ edges can be found in $O(|E|)$ time.

By discarding all unnecessary edges (i.e., edges neither in Υ nor in $\textsc{mwm}(G-H)$), we can assume that G has only $O(\min\{h|E'|+h^2, n_s^2\})$ edges, while still containing $\textsc{mwm}(G)$ and $\textsc{mwm}(G-H)$. This preprocessing requires an extra $O(|E|)$ time for finding $\textsc{mwm}(G)$.

Below, we describe a procedure which, given any bipartite graph D and any node x of D, finds $\textsc{mwm}(D)$ from $\textsc{mwm}(D-\{x\})$ in $O(m_D \log m_D)$ time, where m_D is the number edges of D. Then, starting from $G-H$, we can apply this procedure repeatedly h times to find $\textsc{mwm}(G)$ from $\textsc{mwm}(G-H)$. Since G is assumed to have only $O(\min\{h|E'|+h^2, n_s^2\})$ edges, this process takes $O(h((h|E'|+h^2)\log n_s))$ time. This lemma follows.

Let M and M_x be a maximum weight matching of D and $D-\{x\}$, respectively; denote by S the set of augmenting paths and cycles formed in $M \cup M_x - M \cap M_x$, and let σ be the augmenting path in S starting from x. Note that the augmenting paths and cycles in $S-\{\sigma\}$ cannot improve the matching M_x; otherwise, M_x is not a maximum weight matching of $D-\{x\}$. Thus, we can transform M_x to M using σ. Note that σ is indeed a maximum augmenting path starting from x, which can be found in $O(m_D \log m_D)$ time [5]. □

3 Hierarchical bipartite matching

Throughout this section, let T be a rooted tree as defined in the definition of the hierarchical bipartite matching problem in §1. The root of T is denoted by r. For each node u of T, $w(u)$ and $G_u = (X_u, Y_u, E_u)$ denote the weight and the bipartite graph associated with u, respectively. Furthermore, let $b = \max_{u \in T}\{\min\{|X_u|, |Y_u|\}\}$, and $e = \sum_{u \in T} |E_u|$.

In this section, we describe an algorithm for computing $\textsc{mwm}(G_u)$ for all $u \in T$ in $O(\sqrt{b}w(r)+e)$ time. Our algorithm is based on two crucial observations. One is that for any value x, there are at most $w(r)/x$ graphs with its second maximum edge weight greater than x. The other is that most of these graphs have their total weight dominated by edges incident to a few nodes. For those graphs with a large second maximum edge weight, we compute their maximum weight matchings using a less weight-sensitive algorithm. As there are not many of them, the computation is efficient. For the other graphs, their weights are dominated by the edges incident to a few nodes. Thus, using Lemma 2 and a weight-efficient matching algorithm, we can compute the maximum weight matchings for these graphs efficiently. Details are as follows.

Consider any subset B of nodes of T. Let $\delta = \min_{u \in B} w(u)$. We say that B has a *critical degree* h if for every $u \in B$, u has at most h children with weight at least δ. For any internal node u, let $\textsc{secw}(u) = \text{2nd-}\max\{w(v) \mid v \in C(u)\}$, i.e., the value of the second largest $w(v)$ over all the

children v of u. Lemma 3 below shows the importance of $\textsc{secw}(u)$ and critical degrees. Lemma 3(1) shows that there are not many nodes u with large $\textsc{secw}(u)$; for those nodes u with small $\textsc{secw}(u)$, they should not have a large critical degree, and Lemma 3(2) states that the maximum weight matchings associated with these nodes can be computed efficiently.

Lemma 3.

1. *Let x be any positive number. Let A be the set of nodes u of T with $\textsc{secw}(u) > x$. Then $|A| < w(r)/x$.*
2. *Let B be any set of nodes of T. If B has critical degree h, then we can compute $\textsc{mwm}(G_u)$ for all $u \in B$ in $O((\sqrt{b} + h^3 \log b)w(r) + \sum_{u \in B} |E_u|)$ time.*

Proof. The statement are proved as follows.

Statement 1. Let L be the set of nodes u in T such that (1) $w(u) > x$ and (2) either u is a leaf or $w(v) \leq x$ for all children v of u. Since the subtrees rooted at the nodes of L are disjoint, $w(r) \geq \sum_{u \in L} w(u) > x|L|$. Thus, $|L| < w(r)/x$. Let T' be the tree in T induced by L, i.e., T' contains exactly the nodes of L and the least common ancestor of every two nodes of L. Note that T' has at most $|L|$ leaves and at most $|L|$ internal nodes. On the other hand, every node of A is an internal node of T'; thus, $|A| \leq |L| < w(r)/x$.

Statement 2. For every node $u \in B$, let $H(u)$ be the set of u's children that have a weight at least $\delta = \min_{u \in B} w(u)$ each. Let $L(u)$ be the set of the rest of u's children. Note that $|H(u)| \leq h$ because B has critical degree h. Since the weight of $G_u - H(u)$ is at most $\sum_{x \in L(u)} w(x)$ and $b \geq \min\{|X_u|, |Y_u|\}$, by Theorem 1 we can compute $\textsc{mwm}(G_u - H(u))$ in time

$$O(\sqrt{b} \sum_{x \in L(u)} w(x) + |E_u|). \tag{1}$$

Since $G_u - H(u)$ has at most $\sum_{x \in L(u)} w(x)$ edges and $|H(u)| \leq h$, by Lemma 2, we can compute $\textsc{mwm}(G_u)$ from $\textsc{mwm}(G_u - H(u))$ in time

$$O(|E_u| + (h^2 \sum_{x \in L(u)} w(x) + h^3) \log b) = O(h^3 \log b \sum_{x \in L(u)} w(x) + |E_u|). \tag{2}$$

From Equations (1) and (2), we can compute $\textsc{mwm}(G_u)$ for all $u \in B$ in time

$$O\left(\sum_{u \in B} \left((\sqrt{b} + h^3 \log b) \sum_{x \in L(u)} w(x) + |E_u|\right)\right).$$

Since the subtrees rooted at some node in $\bigcup_{u \in B} L(u)$ are disjoint, $\sum_{u \in B} \sum_{x \in L(u)} w(x) \leq w(r)$. This statement follows. □

We are now ready to compute $\textsc{mwm}(G_u)$ for all nodes u of T. We divide all the nodes in T into two sets: $\Phi = \{u \in T \mid \textsc{secw}(u) > b^3\}$ and $\Pi = \{u \in T \mid \textsc{secw}(u) \leq b^3\}$.

Every node $u \in \Phi$ has $\textsc{secw}(u) > b^3$; by Lemma 3(1), $|\Phi| \leq w(r)/b^3$. Furthermore, by Theorem 1(2), the time for computing $\textsc{mwm}(G_u)$ for all $u \in \Phi$ is $O(\sum_{u \in \Phi}(b^{2.5} \log(bw(u)) + |E_u|))$ $= O(\frac{w(r)}{b^3} b^{2.5} \log w(r) + \sum_{u \in \Phi} |E_u|) = O(w(r)\frac{\log w(r)}{\sqrt{b}} + \sum_{u \in \Phi} |E_u|)$. This time complexity is still far from our goal as $\log w(r)$ may be much larger than $\sqrt{b}$. To improve the time complexity, we first note that using the technique for proving Lemma 2, we can compute $\textsc{mwm}(G_u)$ in time depending only on $\textsc{secw}(u)$. Then, with a better estimation of $\textsc{secw}(u)$, we can reduce the time complexity to $O(w(r) + \sum_{u \in \Phi} |E_u|)$. Details are given in Lemma 4(1).

For Π, we can handle the nodes $u \in \Pi$ with $w(u) > b^3$ easily. For nodes with $w(u) < b^3$, we apply Lemma 3(2) to compute $\text{MWM}(G_u)$. The basic idea is to partition the nodes $u \in \Pi$ into a constant number of sets according to $w(u)$ such that every set has critical degree $b^{\frac{1}{7}}$. This can ensure that the total time to compute all the $\text{MWM}(G_u)$ is $O(\sqrt{b}w(r) + \sum_{u\in\Pi} |E_u|)$. Details are given in Lemma 4(2).

Lemma 4.

1. *We can compute* $\text{MWM}(G_u)$ *for all* $u \in \Phi$ *in* $O(w(r) + \sum_{u\in\Phi} |E_u|)$ *time.*
2. *We can compute* $\text{MWM}(G_u)$ *for all* $u \in \Pi$ *in* $O(\sqrt{b}w(r) + \sum_{u\in\Pi} |E_u|)$ *time.*

Proof. The two statements are proved as follows.

Statement 1. Observe that for any $u \in \Phi$, G_u has at most b^2 edges relevant to the computation of $\text{MWM}(G_u)$, and they can be found in $O(|E_u|)$ time. Let E'_u be this set of edges. Below, we assume that, for every $u \in \Phi$, G_u has only edges in E'_u. Otherwise, it costs $O(\sum_{u\in\Phi} |E_u|)$ extra time to find all E'_u and the assumption holds.

For every $k \geq 1$, let $\Phi_k = \{u \in \Phi \mid 2^{k-1}b^3 < \text{SECW}(u) \leq 2^k b^3\}$. Obviously, the nonempty sets Φ_k form a partition of Φ. Below, we show that for any nonempty Φ_k, we can compute $\text{MWM}(G_u)$ for all $u \in \Phi_k$ in $O\left(w(r)k/2^k + \sum_{u\in\Phi_k} |E'_u| \log b\right)$ time. Thus, the time for computing $\text{MWM}(G_u)$ for all $u \in \Phi$ is $O(\sum_{k\geq 1} w(r)k/2^k + \sum_{u\in\Phi} |E'_u| \log b) = O(w(r) + \sum_{u\in\Phi} |E'_u| \log b) = O(w(r) + (w(r)/b^3)b^2 \log b) = O(w(r))$, and Statement 1 follows.

We now give the details of computing $\text{MWM}(G_u)$ for all $u \in \Phi_k$. Let u' be the child of u where $w(u')$ is the largest over all children of u. Since $\text{SECW}(u) \leq 2^k b^3$, every edge of $G_u - \{u'\}$ has weight at most $2^k b^3$. By Theorem 1(2) and Lemma 2, and the fact $b \geq \min\{|X_u|, |Y_u|\}$, we can find $\text{MWM}(G_u)$ in $O(\sqrt{b}b^2 \log(b2^k b^3) + |E'_u| \log b)$ time. By Lemma 3(1), $|\Phi_k| \leq \frac{w(r)}{2^k b^3}$. Thus, we can compute $\text{MWM}(G_u)$ for all $u \in \Phi_k$ in time

$$\begin{aligned} O\left(\sum_{u\in\Phi_k} \sqrt{b}b^2 \log(b2^k b^3) + |E'_u| \log b\right) &= O\left(\tfrac{w(r)b^{2.5}}{2^k b^3}(k + \log b) + \textstyle\sum_{u\in\Phi_k} |E'_u| \log b\right) \\ &= O\left(w(r)k/2^k + \textstyle\sum_{u\in\Phi_k} |E'_u| \log b\right). \end{aligned}$$

Statement 2. We partition Π as follows. Let Π' be the set of nodes in Π with weight greater than b^3. For any $0 \leq k \leq 20$, let $\Pi_k = \{u \mid u \in \Pi \text{ and } b^{\frac{k}{7}} < w(u) \leq b^{\frac{k+1}{7}}\}$. Obviously, $\Pi = \Pi' \cup \Pi_0 \cup \cdots \Pi_{20}$.

Since $\text{SECW}(u) \leq b^3$ and $w(u) > b^3$ for all nodes u in Π', Π' has critical degree one. By Lemma 3(2), we can compute $\text{MWM}(G_u)$ for all nodes in Π' using $O(\sqrt{b}w(r) + \sum_{u\in\Pi'} |E_u|)$ time.

Each Π_k is handled as follows. For every node $u \in \Pi_k$, u has at most $b^{\frac{1}{7}}$ children with weight at least $b^{\frac{k}{7}}$; otherwise $w(u) > b^{\frac{k+1}{7}}$ and $u \notin \Pi_k$. Thus, Π_k has critical degree $b^{\frac{1}{7}}$. By Lemma 3(2), we can compute $\text{MWM}(G_u)$ for all $u \in \Pi_k$ in $O((\sqrt{b} + b^{\frac{3}{7}} \log b)w(r) + \sum_{u\in\Pi_k} |E_u|)$ $= O(\sqrt{b}w(r) + \sum_{u\in\Pi_k} |E_u|)$ time. In summary, we can compute $\text{MWM}(G_u)$ for all $u \in \Pi$ in $O(\sqrt{b}w(r) + \sum_{u\in\Pi} |E_u|)$ time. □

Theorem 5. *We can compute* $\text{MWM}(G_u)$ *for all nodes* $u \in T$ *in* $O(\sqrt{b}w(r) + e)$ *time.*

Proof. It follows from Lemma 4 and the fact that $T = \Phi \cup \Pi$ and $\sum_{u\in T} |E_u| = e$. □

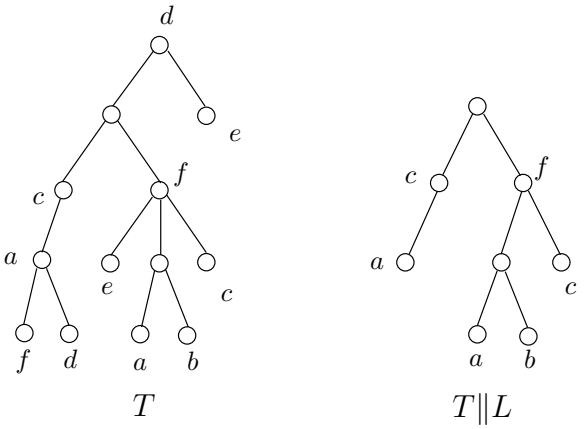

Figure 2: The restricted subtree $T\|L$ with $L = \{a, b, c\}$.

4 Computing maximum agreement subtrees

By generalizing the work of Cole *et al.* [4] on binary evolutionary trees, we can easily derive an algorithm to compute a maximum agreement subtree of two labeled trees. There is, however, a bottleneck of computing the maximum weight matchings of a large number of bipartite graphs with nonconstant degrees. By using our result on the hierarchical bipartite matchings, we can eliminate this bottleneck and obtain the fastest known MAST algorithm. Section 4.1 introduces basics of labeled trees. Section 4.2 uses our results on the hierarchical bipartite matchings to remove the bottleneck in our MAST algorithm. Section 4.3 details our MAST algorithm and analyzes its time complexity. Section 4.4 discusses the generalization of the work of Cole *et al.* [4].

Throughout this section, T_1 and T_2 denote two labeled tress with n nodes and of degree $d \geq 2$. Let $\Delta_{T_1,T_2} = \sum_{u \in T_1} \sum_{v \in T_2} \delta(u, v)$ where $\delta(u, v) = 1$ if nodes u and v are labeled with the same symbol, and 0 otherwise. Also, let Δ denote Δ_{T_1,T_2}.

4.1 Basics

For a rooted tree T and any node u of T, let T^u denote the subtree of T that is rooted at u. For any set L of symbols, the *restricted subtree* of T with respect to L, denoted by $T\|L$, is the subtree of T (1) whose nodes are the nodes with labels from L and the least common ancestors of any two nodes with labels from L and (2) whose edges preserve the ancestor-descendant relationship of T. Figure 2 gives an example. Note that $T\|L$ may contain nodes with labels outside L. For any labeled tree T', let $T\|T'$ denote the restricted subtree of T with respect to the set of symbols used in T'.

A *centroid path decomposition* [4] of a rooted tree T is a partition of its nodes into disjoint paths as follows. For each internal node u in T, let $C(u)$ denote the set of children of u. Among the children of u, one is chosen as the *heavy* child, denoted by $\text{hvy}(u)$, if the subtree of T rooted at $\text{hvy}(u)$ contains the largest number of nodes; the other children of u are *side* children. We call the edge from u to its heavy child a *heavy* edge. A *centroid path* is a maximal path formed by heavy edges; the *root* centroid path is the centroid path that contains the *root* of T. See Figure 3 for an example.

Let $\mathcal{D}(T)$ denote the set of the centroid paths of T. Note that $\mathcal{D}(T)$ can be constructed in

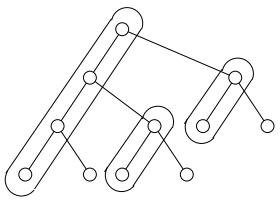

Figure 3: A centroid path decomposition of a rooted tree.

$O(|T|)$ time. For every $P \in \mathcal{D}(T)$, the root of P, denoted $r(P)$, refers to the node on P that is the closest to the root of T, and $\mathcal{A}(P)$ denotes the set of the side children of the nodes on P. For any node u on P, a subtree rooted at some side child of u is called a *side tree* of u, as well as a *side tree* of P. Let SIDE-TREE(P) be the set of side trees of P. Note that for every $R \in$ SIDE-TREE(P), $|R| \leq |T^{r(P)}|/2$.

The following lemma states two useful properties of the centroid path decomposition.

Lemma 6. *Let T_1 and T_2 be two labeled trees.*

1. $\sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} \leq \Delta_{T_1,T_2} \log n$.
2. $\sum_{P \in \mathcal{D}(T_1)} \sqrt{\min(d, |T_1^{r(P)}|)} \Delta_{T_1^{r(P)}, T_2} \leq \sqrt{d} \Delta_{T_1,T_2} \log \frac{2n}{d}$.

Proof. The two statements are proved as follows.

Statement 1. A centroid path P is *attached* to another centroid path P' if the root of P is the child of a node on P'. We define the *level* of a centroid path as follows. The root centroid path has level zero. A centroid path has level i if it is attached to some centroid path with level $i-1$. Note that any subtree attached to a centroid path with level i has size at most $n/2^{i+1}$. Thus, there are at most $\log n$ different levels. Moreover, subtrees attached to centroid paths with the same level are all disjoint.

For any $0 \leq i < \log n$, denote by D_i the set of all centroid paths in $\mathcal{D}(T_1)$ with level i. Then $\sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} = \sum_{0 \leq i < \log n} \sum_{P \in D_i} \Delta_{T_1^{r(P)}, T_2} \leq \log n \sum_{P \in D_i} \Delta_{T_1^{r(P)}, T_2} \leq \Delta_{T_1,T_2} \log n$.

Statement 2. We divide the centroid paths into 2 groups. We first consider the centroid paths on level i where $0 \leq i < \log \frac{2n}{d}$. For any such i,

$$\sum_{P \in D_i} \sqrt{\min\{d, |T_1^{r(P)}|\}} \Delta_{T_1^{r(P)}, T_2} \leq \sum_{P \in D_i} \sqrt{d} \Delta_{T_1^{r(P)}, T_2} \leq \sqrt{d} \Delta_{T_1,T_2}.$$

Thus, $\sum_{0 \leq i < \log \frac{2n}{d}} \sum_{P \in D_i} \sqrt{\min\{d, |T_1^{r(P)}|\}} \Delta_{T_1^{r(P)}, T_2} \leq \sqrt{d} \Delta_{T_1,T_2} \log \frac{2n}{d}$.

Next, we consider the centroid paths on level $\log \frac{2n}{d} + i$ where $i \geq 0$. Note that for a path P on level $\log \frac{2n}{d} + i$, $|T_1^{r(P)}| \leq d/2^{i+1}$. Thus, $\sum_{i \geq 0} \sum_{P \in D_{\log \frac{2n}{d} + i}} \sqrt{\min\{d, |T_1^{r(P)}|\}} \Delta_{T_1^{r(P)}, T_2}$ is at most

$$\sum_{i \geq 0} \sum_{P \in D_{\log \frac{2n}{d} + i}} \sqrt{d/2^{i+1}} \Delta_{T_1^{r(P)}, T_2} \leq \sum_{i \geq 0} \sqrt{d/2^{i+1}} \Delta_{T_1,T_2} = O(\sqrt{d} \Delta_{T_1,T_2}).$$

□

The following notion captures which pairs of nodes of two labeled trees T_1 and T_2 are important. Consider any centroid paths $P \in \mathcal{D}(T_1)$ and $Q \in \mathcal{D}(T_2)$. For any node $x \in P$ or Q, let $\mathcal{L}(x)$ be the set of symbols labeling x and the nodes in the side trees of x. Let $\text{INP}(P, Q)$ be the set of node pairs $(u, v) \in P \times Q$ with $\mathcal{L}(u) \cap \mathcal{L}(v) \neq \emptyset$. Let $\text{INP}(P, T_2) = \bigcup_{Q \in \mathcal{D}(T_2)} \text{INP}(P, Q)$.

4.2 Matchings

As explained later in §4.3, we can easily generalize the dynamic programming approach in [4] to compute $\text{MAST}(T_1, T_2)$ for any two labeled trees T_1 and T_2, but there is a bottleneck of computing the maximum weight matchings of a large number of bipartite graphs with nonconstant degrees. This section uses our results on hierarchical bipartite matchings to remove this bottleneck.

First of all, we identify the bipartite graphs for which maximum weight matchings are required. For any nodes $u \in T_1$ and $v \in T_2$, define G_{uv} as the weighted bipartite graph between $C(u)$ and $C(v)$ where edge (x, y) has weight $\text{MAST}(T_1^x, T_2^y)$. Furthermore, define H_{uv} as the graph constructed from G_{uv} by removing all the zero-weight edges and all the edges adjacent to the heavy child of u or v. Note that the total edge weight of H_{uv} can be significantly smaller than that of G_{uv}. Yet by Lemma 2, we can recover $\text{MWM}(G_{uv})$ from $\text{MWM}(H_{uv})$ efficiently.

This section shows that for any centroid path $P \in \mathcal{D}(T_1)$, we can efficiently compute $\text{MWM}(H_{uv})$ for all $(u, v) \in \text{INP}(P, T_2)$. More precisely, let $\mathcal{TM}_P$ denote the required time; the key result of this section is that $\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P \leq \sqrt{d}\Delta \log \frac{2n}{d}$ (see Lemma 8).

To derive an upper bound on $\mathcal{TM}_P$, we need an estimate of the number of edges in the graphs H_{uv} for all $(u, v) \in \text{INP}(P, T_2)$. For any centroid path $P \in \mathcal{D}(T_1)$, let $\text{tnoe}(P)$ be the total number of edges in the graphs H_{uv} for all $(u, v) \in \text{INP}(P, T_2)$. Furthermore, let $\text{tnoe} = \sum_{P \in \mathcal{D}(T_1)} \text{tnoe}(P)$.

Lemma 7.

1. *For any* $P \in \mathcal{D}(T_1)$, $\text{tnoe}(P) = O\left(\sum_{w \in \mathcal{A}(P)} |T_2 \| T_1^w| \log \frac{2|T_2 \| T_1^{r(P)}|}{|T_2 \| T_1^w|}\right)$.
2. $\text{tnoe} = O(\Delta \log n)$.

Proof. The two statements are proved as follows.

Statement 1. Let r be the root of P. By definition, every edge in H_{uv} for any $(u, v) \in \text{INP}(P, T_2)$ corresponds to a pair of side trees (Υ, Π) where $\Upsilon \in \text{SIDE-TREE}(P)$ and $\Pi \in \text{SIDE-TREE}(Q)$ for some $Q \in \mathcal{D}(T_2 \| T_1^r)$ such that Υ and Π contain some common labels. We call (Υ, Π) an *intersecting* side tree pair. Thus, $\text{tnoe}(P)$ is at most the total number of intersecting side tree pairs in $\text{SIDE-TREE}(P) \times \bigcup\{\text{SIDE-TREE}(Q) \mid Q \in \mathcal{D}(T_2 \| T_1^r)\}$.

To simplify our discussion, let $R = T_2 \| T_1^r$ and $\text{SIDE-TREE}(R) = \bigcup\{\text{SIDE-TREE}(Q) \mid Q \in \mathcal{D}(R)\}$. Consider any node $w \in \mathcal{A}(P)$. T_1^w is a side tree in $\text{SIDE-TREE}(P)$. Let R_w be $T_2 \| T_1^w$. Note that each path in R_w starting from a node x to its descendant y corresponds to a simple path Q_{xy} in R from x to y. Let $1^{st}(x, y)$ be the node on Q_{xy} which is the child of x. By the definition of side trees, among all the side trees in $\text{SIDE-TREE}(R)$, at most $\log |R^{1^{st}(x,y)}| + 1$ have roots on Q_{xy}.

For all side trees $R^v \in \text{SIDE-TREE}(R)$, (T_1^w, R^v) is an intersecting side tree pair if and only if either (1) v is a node on the path from the root of R to the root R_w; or (2) v is a node on some path Q_{xy} on R where (x, y) is an edge in R_w. The number of side trees $R^v \in \text{SIDE-TREE}(R)$ in case (1) is less than $\log |R|$. The number of side trees $R^v \in \text{SIDE-TREE}(R)$ in case (2) is less than $\sum_{(x,y) \in R_w} \left(\log |R^{1^{st}(x,y)}| + 1\right)$. Let $\text{SUM}(R_w)$ denote $\sum_{(x,y) \in R_w} \log |R^{1^{st}(x,y)}|$. Below we prove

$\textsc{sum}(R_w) = O\left(|R_w| \log \frac{2|R|}{|R_w|}\right)$. In total, $\text{tnoe}(P) = O\left(\sum_{w \in \mathcal{A}(P)} \left\{|R_w| \log \frac{2|R|}{|R_w|}\right\}\right)$, as claimed in this statement.

It remains to prove $\textsc{sum}(R_w) = O\left(|R_w| \log \frac{2|R|}{|R_w|}\right)$. For any leaf y of R, let $p(y)$ be the maximal path in R ending at y such that every node on $p(y)$ has at most one child; denote $r_p(y)$ as the root of $p(y)$. Let $Z_{R_w} = \{p(y) \mid y \text{ is a leaf of } R_w\}$. As $\{R^{1^{st}(r_p(y),y)} \mid p(y) \in Z_{R_w}\}$ is a set of disjoint subtrees of R, $\sum_{p(y) \in Z_{R_w}} |R^{1^{st}(r_p(y),y)}| \leq |R|$. Note that $|Z_{R_w}| \leq |R_w|$. Thus, $\sum_{p(y) \in Z_{R_w}} \log |R^{1^{st}(r_p(y),y)}| \leq |R_w| \log \frac{2|R|}{|R_w|}$.[1] Let $\hat{R}_w$ be the tree obtained by removing all the paths in Z_{R_w}. We have $\textsc{sum}(R_w) = |R_w| \log \frac{2|R|}{|R_w|} + \textsc{sum}(\hat{R}_w)$. Note that $\hat{R}_w$ contains at most half the leaves of R_w. Hence, $\textsc{sum}(R_w) = O\left(|R_w| \log \frac{2|R|}{|R_w|}\right)$.

Statement 2. By Statement 1, tnoe is in the order of

$$
\begin{aligned}
& \sum_{P \in \mathcal{D}(T_1)} \left[\sum_{w \in \mathcal{A}(P)} |T_2\|T_1^w| \log \frac{2|T_2\|T_1^{r(P)}|}{|T_2\|T_1^w|} \right] \\
\leq & \sum_{P \in \mathcal{D}(T_1)} \left[\sum_{w \in \mathcal{A}(P)} \Delta_{T_1^w, T_2} \log \frac{2|T_2\|T_1^{r(P)}|}{\log |T_2\|T_1^w|} \right] \\
= & \sum_{P \in \mathcal{D}(T_1)} \left[\sum_{w \in \mathcal{A}(P)} \Delta_{T_1^w, T_2} (1 + \log |T_2\|T_1^{r(P)}| - \log |T_2\|T_1^w|) \right] \\
\leq & \sum_{P \in \mathcal{D}(T_1)} \left[\Delta_{T_1^{r(P)}, T_2} + \Delta_{T_1^{r(P)}, T_2} \log |T_2\|T_1^{r(P)}| - \sum_{w \in \mathcal{A}(P)} \Delta_{T_1^w, T_2} \log |T_2\|T_1^w| \right] \\
\leq & \sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} + \sum_{P \in \mathcal{D}(T_1)} \left[\Delta_{T_1^{r(P)}, T_2} \log |T_2\|T_1^{r(P)}| - \sum_{w \in \mathcal{A}(P)} \Delta_{T_1^w, T_2} \log |T_2\|T_1^w| \right] \\
= & \sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} + \Delta_{T_1^{r_o}, T_2} \log |T_2\|T_1^{r_o}|, \text{ where } r_o \text{ is the root of } T_1 \\
\leq & \sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} + \Delta_{T_1, T_2} \log |T_2| \\
\leq & \Delta_{T_1, T_2} \log |T_2| + \Delta_{T_1, T_2} \log |T_2| \qquad \text{by Lemma 6(1)} \\
= & 2\Delta \log n.
\end{aligned}
$$

□

We proceed to detail the computing of $\textsc{mwm}(H_{uv})$ for all $(u, v) \in \bigcup_{P \in \mathcal{D}(T_1)} \textsc{inp}(P, T_2)$. A bipartite graph is *nontrivial* if both node sets have at least two nodes. Computing $\textsc{mwm}(H_{uv})$ for all trivial H_{uv} takes only linear time, i.e., $O(\text{tnoe}) = O(\Delta \log n)$ time. Thus, we focus on those nontrivial H_{uv}.

Consider any centroid path P in $\mathcal{D}(T_1)$ and fix a node u of P. Let $\mathcal{H}_u$ be the set of all nontrivial graphs H_{uv} where $(u, v) \in \textsc{inp}(P, T_2)$. Let $\mathcal{TM}_u$ be the time for finding $\textsc{mwm}(H_{uv})$ for all the graphs in $\mathcal{H}_u$. Let $\text{tnoe}(u)$ be the number of edges of all the graphs in $\mathcal{H}_u$. In the next lemma, we first derive an upper bound of $\mathcal{TM}_u$, and then we show $\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P = O(\sqrt{d}\Delta \log \frac{2n}{d})$.

[1] This follows from the fact that for any sequence of positive numbers $a_1, a_2, \ldots, a_k$ with the sum equal to s, $\sum_{i=1}^{k} \log a_i \leq \ell \log \frac{2s}{\ell}$, where $k \leq \ell \leq s$.

Lemma 8.

1. $\mathcal{TM}_u = O(\sqrt{\min(d, |T_1^u|)}\Delta_{S_u,T_2} + \text{tnoe}(u))$, *where* S_u *is the set of side trees of* u *in* T_1 *and* $\Delta_{S_u,T_2} = \sum_{\Upsilon \in S_u} \Delta_{\Upsilon,T_2}$.
2. $\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P = O(\sqrt{d}\Delta \log \frac{2n}{d})$.

Proof. The two statements are proved as follows.

Statement 1. Let B_u be the set of labels used in the side trees in S_u. First, we show that for all $v \in T_2 \| B_u$, $\text{MWM}(H_{uv})$ can be computed in $O\Big(\sqrt{\min(d, |T_1^u|)}\Delta_{S_u,T_2} + \text{tnoe}(u)\Big)$ time. Second, we recover $\text{MWM}(H_{uv})$ for all nontrivial H_{uv} where $(u, v) \in \text{INP}(P, T_2)$ in $O(\Delta_{S_u,T_2})$ time. Then this statement follows.

To compute $\text{MWM}(H_{uv})$ for all $v \in T_2 \| B_u$, we apply the hierarchical bipartite matching algorithm of §3. Let $T = T_2 \| B_u$. For every node $v \in T$, we associate with v the bipartite graph H_{uv} and let $w(v) = \Delta_{S_u,T_2^v}$. Observe that $w(v) = \Delta_{S_u,T_2^v} \geq \sum_{x \in C(v)} \Delta_{S_u,T_2^x} = \sum_{x \in C(v)} w(x)$. In addition, for every node $x \in C(v)$, the total weight of all the edges incident to x in H_{uv} is at most $w(x) = \Delta_{S_u,T_2^x}$. Hence, T and the associated bipartite graphs H_{uv} satisfy the conditions for the hierarchical bipartite matching problem. For the time complexity, note that, for every $v \in T$, the two node sets of H_{uv} have size bounded by d and $\min(d, |T_1^u|)$, respectively. Thus, by Theorem 5, we can find $\text{MWM}(H_{uv})$ for all nodes v of $T = T_2 \| B_u$ in $O(\sqrt{\min(d, |T_1^u|)}\Delta_{S_u,T_2} + \text{tnoe}(u))$ time.

Next, we show how to recover $\text{MWM}(H_{uv})$ for all $v \in L$, where L denotes the set of nodes v of T_2 such that $(u, v) \in \text{INP}(P, T_2)$ and H_{uv} is nontrivial. Note that every node x of $T_2 \| B_u$ is also a node in T_2 and every edge (x, y) of $T_2 \| B_u$ corresponds to a path in T_2. Also observe that every node $v \in L$ must be a node in $T_2 \| B_u$; otherwise, v lies on a path corresponding to an edge (x, y) of $T_2 \| B_u$, and H_{uv} contains a singleton node set and is trivial because $v \notin L$. Therefore, we can compute $\text{MWM}(H_{uv})$ for all $v \in L$ by traversing $T_2 \| B_u$ once using $O(|(T_2 \| B_u)|) = O(\Delta_{S_u,T_2})$ time.

Statement 2. By Statement 1,

$$\begin{aligned}
\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P &= O\left(\sum_{P \in \mathcal{D}(T_1)} \sum_{u \in P} \mathcal{TM}_u\right) \\
&= O\left(\sum_{P \in \mathcal{D}(T_1)} \sum_{u \in P} (\sqrt{\min(d, |T_1^u|)}\Delta_{S_u,T_2} + \text{tnoe}(u))\right) \\
&= O\left(\text{tnoe} + \sum_{P \in \mathcal{D}(T_1)} \sum_{u \in P} \sqrt{\min(d, |T_1^u|)}\Delta_{S_u,T_2}\right) \\
&= O\left(\text{tnoe} + \sum_{P \in \mathcal{D}(T_1)} \sqrt{\min(d, |T_1^{r(P)}|)}\Delta_{T_1^{r(P)},T_2}\right) \\
&= O\left(\text{tnoe} + \sqrt{d}\Delta_{T_1,T_2} \log \frac{2n}{d}\right) \qquad \text{by Lemma 6} \\
&= O\left(\sqrt{d}\Delta \log \frac{2n}{d}\right) \qquad \text{by Lemma 7}
\end{aligned}$$

□

4.3 The MAST algorithm

Our algorithm is based on the following recurrence, which generalizes the one given in [7] to handle labeled trees.

$$\textsc{mast}(T_1^u, T_2^v) = \max \begin{cases} \max\{\textsc{mast}(T_1^u, T_2^x) \mid x \in C(v)\}, \\ \max\{\textsc{mast}(T_1^x, T_2^v) \mid x \in C(u)\}, \\ \|\textsc{mwm}(G_{uv})\| \text{ if } u \text{ and } v \text{ are both unlabeled}, \\ \|\textsc{mwm}(G_{uv})\| + 1 \text{ for } u, v \text{ labeled with the same symbol}, \end{cases} \tag{3}$$

where $\|\textsc{mwm}(G_{uv})\|$ denotes the total weight of the matching.

Equation (3) suggests a bottom-up dynamic programming approach to computing $\textsc{mast}(T_1, T_2)$. The following lemma generalizes the technique of Cole *et al.* [4] for speeding up the dynamic programming. Basically, it states that the time complexity is dominated by the time for finding maximum weight matchings of some graphs H_{uv}.

Lemma 9. *Let $P \in \mathcal{D}(T_1)$ be a centroid path and $r = r(P)$. Given the values $\textsc{mast}(T_1^u, (T_2 \| T_1^u)^v)$ for all nodes $u \in \mathcal{A}(P)$ and $v \in T_2 \| T_1^u$, we can compute $\textsc{mast}(T_1^r, (T_2 \| T_1^r)^v)$ for all $v \in T_2 \| T_1^r$ in $O\left(\left(\gamma(T_1^r) - \sum_{u \in \mathcal{A}(P)} \gamma(T_1^u) + \Delta_{T_1^r, T_2}\right) \log d + \mathcal{TM}_P\right)$ time, where $\gamma(R)$ denotes $\Delta_{R,T_2} \log |(T_2 \| R)|$.*

Cole *et al.* [4] proved Lemma 9 for the special case where T_1 and T_2 are binary evolutionary trees. For a better flow of discussion, we postpone the proof of Lemma 9 to §4.4. Here, Lemma 9 immediately suggests that $\textsc{mast}(T_1, T_2)$ can be computed in a bottom-up fashion as follows:

- Step 1. Let $\prec$ denote the ordering on $\mathcal{D}(T)$ where $P_1 \prec P_2$ if the root of P_1 is a descendant of the root P_2.
- Step 2. For every $P \in \mathcal{D}(T_1)$ in increasing order according to $\prec$, let r denote the root of P; apply Lemma 9 to find $(T_1^r, (T_2 \| T_1^r)^v)$ for every node $v \in T_2 \| T_1^r$.

The above algorithm at the end computes $\textsc{mast}(T_1^{r(P_o)}, (T_2 \| T_1^{r(P_o)}))$, where P_o is root centroid path of T_1. Since $r(P_o)$ is also the root of T, we have $T_1^{r(P_o)} = T_1$ and $\textsc{mast}(T_1^{r(P_o)}, (T_2 \| T_1^{r(P_o)})) = \textsc{mast}(T_1, T_2)$. As stated in the following lemma, the running time is dominated by the time for computing the maximum weight matchings, i.e., $\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P$.

Lemma 10. *We can compute $\textsc{mast}(T_1, T_2)$ in $O(\Delta \log n \log d + \sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P)$ time, where $\Delta = \Delta_{T_1, T_2}$.*

Proof. To derive the time for computing $\textsc{mast}(T_1, T_2)$, we simply sum the time bound stated in Lemma 9 over all centroid paths of T_1. Observe that

$$\begin{aligned} \sum_{P \in \mathcal{D}(T_1)} \left[\gamma(T_1^{r(P)}) - \sum_{u \in \mathcal{A}(P)} \gamma(T_1^u)\right] &= \gamma(T_1^{r_o}), \text{ where } r_o \text{ is the root of of } T_1 \\ &= \Delta_{T_1, T_2} \log |T_2 \| T_1| \\ &= \Delta_{T_1, T_2} \log |T_2| \\ &= \Delta \log n. \end{aligned}$$

Thus, we can compute $\textsc{mast}(T_1, T_2)$ in $O(\Delta \log n \log d + \sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P + \sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} \log d)$ time. By Lemma 6, $\sum_{P \in \mathcal{D}(T_1)} \Delta_{T_1^{r(P)}, T_2} \leq \Delta \log n$. Thus, this lemma follows. □

Theorem 11. $\text{MAST}(T_1, T_2)$ *can be computed in* $O(\sqrt{d}\Delta_{T_1,T_2} \log \frac{2n}{d})$ *time.*

Proof. By Lemma 8, $\sum_{P \in \mathcal{D}(T_1)} \mathcal{TM}_P = O(\sqrt{d}\Delta \log \frac{2n}{d})$. Thus, by Lemma 10, $\text{MAST}(T_1, T_2)$ can be computed in $O(\Delta(\log n \log d + \sqrt{d} \log \frac{2n}{d}))$ time. Since $\log n \log d \leq \sqrt{d} \log \frac{2n}{d}$, this theorem follows. □

4.4 Proof of Lemma 9

This section provides the details for adapting the techniques of Cole *et al.* [4] to prove Lemma 9. Consider any centroid path $P \in \mathcal{D}(T_1)$. Let r be the root of P. Lemma 9 states that if we are given, for every $u \in \mathcal{A}(P)$,

$$\text{MAST}(T_1^u, (T_2 \| T_1^u)^v) \text{ for all } v \in T_2 \| T_1^u,$$

then we can compute

$$\text{MAST}(T_1^r, (T_2 \| T_1^r)^v) \text{ for all } v \in T_2 \| T_1^r \tag{4}$$

in $O\left((\gamma(T_1^r) - \sum_{u \in \mathcal{A}(P)} \gamma(T_1^u) + \Delta_{T_1^r, T_2}) \log d + \mathcal{TM}_P\right)$ time.

The centroid paths in $\mathcal{D}(T_2 \| T_1^r)$ partition the set of nodes of $T_2 \| T_1^r$ and define an ordering on the nodes of $T_2 \| T_1^r$. Precisely, the set of values in Equation (4) are partitioned into the following sets

$$\{\text{MAST}(T_1^r, (T_2 \| T_1^r)^v) \mid v \in Q\} \text{ where } Q \in \mathcal{D}(T_2 \| T_1^r).$$

We focus on computing $\{\text{MAST}(T_1^r, (T_2 \| T_1^r)^v) \mid v \in Q\}$ for each $Q \in \mathcal{D}(T_2 \| T_1^r)$. Cole *et al.* [4] dealt with the special case where T_1 and T_2 are binary evolutionary trees. They introduced the *maximum agreement matching* (MAM) problem and showed that $\{\text{MAST}(T_1^r, (T_2 \| T_1^r)^v) \mid v \in Q\}$ can be computed by solving the MAM problem on some weighted bipartite multigraph. In [22], Przytycka observed that this technique can be generalized to evolutionary trees of arbitrary degrees; basically, it suffices to use a more complicated bipartite multigraph. We observe that this can be further generalized to labeled trees with arbitrary degrees by adding more edges to the multigraph.

In the rest of this section, we define the maximum agreement matching problem and the weighted bipartite multigraph $\mathcal{G}_{PQ}$ for handling labeled trees with general degrees.

The maximum agreement matching problem. Let $\mathcal{G} = (X, Y, E)$ be a weighted bipartite multigraph. Suppose that $X = \{u_1, u_2, \ldots, u_p\}$, $Y = \{v_1, v_2, \ldots, v_q\}$, and every pair of nodes are connected by at most four edges. Every edge is colored by either *gray, green, red,* or *white.* We say that edge (u_i, v_j) is *below* edge (u_k, v_ℓ) if $i < k$ and $j < \ell$; and that (u_i, v_j) *crosses* (u_k, v_ℓ) if $i < k$ and $j > \ell$. A matching of $\mathcal{G}$ is an *agreement* matching if it satisfies all the following properties:

- No white edge crosses another white edge.
- There is at most one gray edge. If a gray edge is present, it must be below all the white edges.
- There are at most one pair of red and green edges. If such a pair is present, then this pair of edges are below all white edges, and the red edge crosses the green edge.
- A gray edge cannot coexist with a pair of red and green edges.

The *weight* of an agreement matching of $\mathcal{G}$ is the total weight of the edges in the matching. A *maximum agreement matching* is one with the maximum weight, and we denote this weight as $\text{MAM}(\mathcal{G})$.

For any nodes $u_i \in X$ and $v_j \in Y$, let $\mathcal{G}(u_i, v_j)$ denote the subgraph of $\mathcal{G}$ induced by the nodes $u_i, u_{i+1}, \ldots, u_p$ and $v_j, v_{j+1}, \ldots, v_q$. The maximum agreement matching problem asks for $\text{MAM}(\mathcal{G}(u_i, v_j))$ for all pairs of (u_i, v_j) such that either (1) $u_i = u_1$ and v_j is adjacent to some edges of $\mathcal{G}$; or (2) $v_j = v_1$ and u_i is adjacent to some edges of $\mathcal{G}$.

The weighted bipartite multigraph $\mathcal{G}_{PQ}$. Roughly speaking, $\mathcal{G}_{PQ}$ is constructed by adding suitable colored edges between P and Q. Our aim is that by solving the MAM problem on $\mathcal{G}_{PQ}$, all the values in Equation (4) are found automatically.

First, we define a new graph H'_{uv} from G_{uv} and H_{uv} as follows. H'_{uv} has all the edges of H_{uv}, as well as some other edges from G_{uv}. Among all the edges of G_{uv} adjacent to $\text{hvy}(u)$, we add into H'_{uv} those edges $(\text{hvy}(u), y)$ where y is adjacent to some edges of H_{uv}. Among the rest of the edges adjacent to $\text{hvy}(u)$, we add into H'_{uv} the one with the heaviest weight. Similarly, among all edges adjacent to $\text{hvy}(v)$, we choose some edges to add into H'_{uv}.

We are now ready to define $\mathcal{G}_{PQ}$. Suppose that $P = (u_1, u_2, \ldots, u_p)$ and $Q = (v_1, v_2, \ldots, v_q)$. There is one or more edges between nodes u_i and v_j if and only if $(u_i, v_j) \in \text{INP}(P, Q)$. The number, color, and weight of edges between u_i and v_j are determined in the three cases below. Let $\text{MAX}_R = \max\{\text{MAST}(T_1^{u_i}, \Gamma) \mid \Gamma \text{ is a side tree of } v_j\}$. Let $\text{MAX}_L = \max\{\text{MAST}(\Gamma, T_2^{v_j}) \mid \Gamma \text{ is a side tree of } u_i\}$.

Case 1: u_i and v_j are both unlabeled. There are a white edge, a gray edge, a green edge and a red edge connecting u_i and v_j, with weights $\|\text{MWM}(H_{u_i v_j})\|$, $\|\text{MWM}(H'_{u_i v_j})\|$, MAX_R, and MAX_L, respectively.

Case 2: u_i and v_j are labeled by the same symbol z. There are a white edge and a gray edge connecting them. The weight of the white edge is $\|\text{MWM}(H_{u_i v_j})\| + \mu(z)$. The weight of the gray edge equals the maximum of $\|\text{MWM}(H'_{u_i v_j})\| + \mu(z)$, MAX_R, and MAX_L.

Case 3: either u_i and v_j are labeled by different symbols, or only one of them is labeled. There is only one gray edge connecting them. Its weight equals the larger of MAX_R and MAX_L.

Note that when the input is evolutionary trees, $\mathcal{G}_{PQ}$ is reduced to the multigraph defined in [22], in which most of the edges are from Case 1, and there are edges (u_i, v_j) from Cases 2 and 3 only when u_i and v_j are leaves. For labeled trees, we simply add extra edges in Cases 2 and 3 when u_i or v_j is a labeled internal node. By construction, we have the following fact.

Fact 12. *For any $u_i \in P$ and $v_j \in Q$, $\text{MAM}(\mathcal{G}_{PQ}(u_i, v_j)) = \text{MAST}(T_1^{u_i}, (T_2 \| T_1^r)^{v_j})$. Thus, solving the* MAM *problem on $\mathcal{G}_{PQ}$ gives $\text{MAST}(T_1^r, (T_2 \| T_1^r)^v)$ for all $v \in Q$.*

Using the techniques of Cole *et al.* [4, 22], we can construct $\mathcal{G}_{PQ}$ for all $Q \in \mathcal{D}(T_2 \| T_1^r)$ and solve the corresponding MAM problems in $O\left(\left(\gamma(T_1^r) - \sum_{u \in \mathcal{A}(P)} \gamma(T_1^u) + \Delta_{T_1^r, T_2}\right) \log d + \mathcal{TM}_P\right)$ total time. Therefore, Lemma 9 follows.

Acknowledgments

We wish to thank anonymous referees for extremely helpful suggestions.

References

[1] S. Abiteboul. On views and XML. In *Proceedings of the ACM Symposium on Principle of Database Systems*, pages 1–9, 1999.

[2] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5):906–933, 1994.

[3] M. J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8:106–112, 1987.

[4] R. Cole, M. Farach, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 2000. To appear.

[5] T. H. Cormen, C. L. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[6] M. Farach, T. M. Przytycka, and M. Thorup. Computing the agreement of trees with bounded degrees. In P. Spirakis, editor, *Lecture Notes in Computer Science 979: Proceedings of the 3rd Annual European Symposium on Algorithms*, pages 381–393. Springer-Verlag, New York, NY, 1995.

[7] M. Farach and M. Thorup. Sparse dynamic programming for evolutionary-tree comparison. *SIAM Journal on Computing*, 26(1):210–230, 1997.

[8] C. R. Finden and A. D. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.

[9] J. Friedman. Expressing logical formulas in natural languages. In J. Groenendijk, T. Janssen, and M. Stokhof, editors, *Formal methods in the study of language*, pages 113–130. Mathmatical Centre, Amsterdam, 1981.

[10] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[11] A. Gupta and N. Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.

[12] D. Gusfield, C. Martel, and D. Fernández-Baca. Fast algorithms for bipartite network flow. *SIAM Journal on Computing*, 16(2):237–251, 1987.

[13] D. M. Hillis, C. Moritz, and B. K. Mable, editors. *Molecular Systematics*. Sinauer Associates, Sunderland, Ma, 2nd edition, 1996.

[14] M. Y. Kao. Tree contractions and evolutionary trees. *SIAM Journal on Computing*, 27(6):1592–1616, December 1998.

[15] M. Y. Kao, T. W. Lam, W. K. Sung, and H. F. Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal on Computing*, 2000. To appear.

[16] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 214–222, 1991.

[17] P. Kilpeläinen and H. Mannila. Grammatical tree matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Lecture Notes in Computer Science 644: Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*, pages 162–174. Springer-Verlag, New York, NY, 1992.

[18] B. Kimia, A. Tannenbaum, and S. W. Zucker. Shapes, shocks, and deformations, I. *International Journal of Computer Vision*, pages 189–224, 1995.

[19] S. Y. Le, J. Owens, R. Nussinov, J. H. Chen, B. Shapiro, and J. V. Maizel. RNA secondary structures: comparison and determination of frequently recurring substructures by consensus. *Computer Application in Bioscience*, 5:205–210, 1989.

[20] H. Mannila and K. J. Räihä. On query languages for the p-string data model. In H. Kangassalo, S. Ohsuga, and H. Jaakkola, editors, *Information Modelling and Knowledge Bases*, pages 469–482. IOS Press, Amsterdam, 1990.

[21] P. Materna, P. Sgall, and Z. Hajicova. Linguistic constructions in transparent intensional logic. *Prague Bulletin on Mathematical Linguistics*, pages 27–32, 1985.

[22] T. Przytycka. Sparse dynamic programming for maximum agreement subtree problem. In B. Mirkin, F. R. McMorris, F. S. Roberts, and A. Rzhetsky, editors, *Mathematical Hierarchies and Biology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 249–264. American Mathematical Society, Providence, RI, 1997.

[23] B. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in Bioscience*, pages 309–318, 1990.

[24] M. Steel and T. J. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48:77–82, 1993.

[25] Y. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki. Recognition of largest common structural fragment among a variety of chemical structures. *Analytical Science*, pages 23–28, 1987.

[26] T. Tokuyama and J. Nakano. Efficient algorithms for the Hitchcock transportation problems. *SIAM Journal on Computing*, 24(3):563–578, 1995.

[27] T. Tokuyama and J. Nakano. Geometric algorithms for the minimum cost assignment problem. *Random Structures and Algorithms*, 6:393–406, 1995.