Algorithmica (2007) 48: 233–248
DOI: 10.1007/s00453-007-0080-9

Algorithmica

Polynomial-Time Algorithms for the Ordered Maximum Agreement Subtree Problem[1]

Anders Dessmark,[2] Jesper Jansson,[3] Andrzej Lingas,[2] and Eva-Marta Lundell[2]

Abstract. For a set of rooted, unordered, distinctly leaf-labeled trees, the NP-hard maximum agreement subtree problem (MAST) asks for a tree contained (up to isomorphism or homeomorphism) in all of the input trees with as many labeled leaves as possible. We study the *ordered* variants of MAST where the trees are uniformly or non-uniformly ordered. We provide the first known polynomial-time algorithms for the uniformly and non-uniformly ordered homeomorphic variants as well as the uniformly and non-uniformly ordered isomorphic variants of MAST. Our algorithms run in time $O(kn^3)$, $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$, $O(kn^3)$, and $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$, respectively, where n is the number of leaf labels and k is the number of input trees.

Key Words. Evolutionary tree, Maximum agreement subtree, Ordered tree, Algorithm, Time complexity.

1. Introduction. The basic combinatorial problem of finding a *longest common subsequence* (LCS) for a set of sequences and the well-known problem of finding a *maximum agreement subtree* (MAST) for a set of trees with distinctly labeled leaves fall in the general category of problems of finding a *largest common sub-object* for an input set of combinatorial objects. In [7] Fellows et al. in particular studied the largest common sub-object problem constrained to so-called p-sequences, i.e., sequences where each element occurs at most once. In this paper we consider a natural generalization of the largest common sub-object problem for p-sequences in which the objects are allowed to be arbitrary rooted, ordered trees with distinctly labeled leaves (note that a p-sequence is equivalent to a rooted, ordered, distinctly leaf-labeled star tree). Since this problem can be also regarded as a restriction of the MAST problem where tree ordering is required, we term it *the ordered maximum agreement subtree problem*.

For an extensive literature and motivation for the LCS and MAST problems, the reader is referred to [3], [16], [19] and [2], [4], [5], [6], [9], [10], [11], [14], [15], [17], [18], respectively. Indeed, the NP-hardness [2] and approximation NP-hardness [4], [10], [11] of the general MAST problem is one of the motivations for studying its ordered variants in this paper.

---

[1] A preliminary version of this article has appeared in *Proceedings of the* 15*th Annual Symposium on Combinatorial Pattern Matching* (*CPM* 2004), volume 3109 of Lecture Notes in Computer Science, pages 220–229, Springer-Verlag, Berlin, 2004.

[2] Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. {Anders.Dessmark, Andrzej.Lingas,Eva-Marta.Lundell}@cs.lth.se.

[3] Department of Computer Science and Communication Engineering, Kyushu University, Japan. jj@tcslab.csce.kyushu-u.ac.jp. Supported in part by JSPS (Japan Society for the Promotion of Science).

Received November 10, 2004; revised March 17, 2005, and January 3, 2006. Communicated by T. Jiang. Online publication June 25, 2007.

1.1. *Variants of the MAST Problem*. A tree whose leaves are labeled by elements belonging to a finite set S so that no two leaves have the same label is said to be *distinctly leaf-labeled by* S. Throughout this paper, each leaf in such a tree is identified with its corresponding element in S. Let T be a rooted tree distinctly leaf-labeled by a given finite set S. For any subset S' of S, $T \mid S'$ denotes the tree obtained by first deleting from T all leaves which are not in S' and all internal nodes without any descendants in S' along with their incident edges, and then contracting every edge between a node having just one child and its child. Similarly, $T \parallel S'$ denotes the tree obtained by first deleting from T all leaves which are not in S' and all internal nodes without any descendants in S' along with their incident edges, and then if the resulting root only has one child, contracting every edge on the path from the root to its first descendant with more than one child.

In *the maximum homeomorphic agreement subtree problem* (MHT), the input is a finite set S and a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of rooted, unordered trees, where each $T_x \in \mathcal{T}$ is distinctly leaf-labeled by S and no $T_x \in \mathcal{T}$ has a node of degree 1, and the goal is to find a subset S' of S of maximum cardinality such that $T_1 \mid S' = \cdots = T_k \mid S'$. In *the maximum isomorphic agreement subtree problem* (MIT), the input is a finite set S and a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of rooted, unordered trees, where each $T_x \in \mathcal{T}$ is distinctly leaf-labeled by S, and the goal is to find a subset S' of S of maximum cardinality such that $T_1 \parallel S' = \cdots = T_k \parallel S'$.

An *ordered* tree is a rooted tree in which the left-to-right order of the children of each node is significant. The *leaf ordering* of an ordered, leaf-labeled tree is the sequence of labels obtained by scanning its leaves from left to right. A set $\mathcal{T}$ of ordered trees distinctly leaf-labeled by S is said to be *uniformly ordered* if all trees in $\mathcal{T}$ have the same leaf ordering.

We study the following four ordered variants of MHT and MIT:

- The ordered maximum homeomorphic agreement subtree problem (OMHT).
- The ordered maximum isomorphic agreement subtree problem (OMIT).
- The uniformly ordered maximum homeomorphic agreement subtree problem (UOMHT).
- The uniformly ordered maximum isomorphic agreement subtree problem (UOMIT).

OMHT and OMIT are defined in the same way as MHT and MIT except that $\mathcal{T}$ is required to be a set of *ordered* trees. UOMHT and UOMIT are the special cases of OMHT and OMIT in which $\mathcal{T}$ is required to be uniformly ordered. Note that OMHT and OMIT generalize the largest common sub-object problem for p-sequences studied by Fellows et al. in [7].

From here on, n and k denote the cardinalities of S and $\mathcal{T}$, respectively.

1.2. *Motivation*. In certain evolutionary tree construction situations, one can determine or accurately estimate the leaf ordering of a planar embedding of the true tree by taking into account other kinds of data such as the geographical distributions of the species or data based on some measurable quantitative characteristics (average life span, size,

etc.).[4] The ordered variants of MHT and MIT might also arise in graphical representation of evolutionary trees where additional restrictions are placed on the leaves (e.g., that they must be ordered alphabetically) for ease of presentation.

In the context of the approximation NP-hardness of MHT and MIT (see [4], [10], [11] and Section 1.3 below), their ordered restrictions are of theoretical interest in their own rights. Does the leaf ordering restriction make the problems computationally feasible? In [9] Gąsieniec et al. proved that an analogous ordering restriction on an NP-hard optimization problem occurring in the construction of evolutionary trees admits a polynomial-time algorithmic solution.[5] Moreover, Jiang et al. [13] have shown that the alignment of trees problem, defined in [13], is MAX SNP-hard for unordered trees already when one of the two input trees has unrestricted degree, but can be solved in polynomial time for two ordered trees of unrestricted degree. Our results on the ordered variants of MHT and MIT further confirm the power of ordering.

1.3. *Related Results.* Fellows et al. studied the longest common subsequence problem for p-sequences (among many other problems) in [7], and claimed that they could solve this problem for k p-sequences with n symbols in $O(kn(k+\log n))$ time. However, their algorithm seems to fail already for $x_1 = 1234$ and $x_2 = 1342$, producing either 12 or 34 instead of 134. Therefore, we were forced to use a weaker upper time-bound for this problem, given in Lemma 10 in this paper, in order to derive one of our main results.

Many results for the unordered MAST problem have been published previously. Steel and Warnow [17] presented the first exact polynomial-time algorithms for MHT and its unrooted counterpart UMHT[6] for the case $k = 2$. The currently fastest algorithm for MHT with $k = 2$, due to Kao et al. [14], runs in $O(\sqrt{D}\, n \log(2n/D))$ time, where D is the maximum degree of the two input trees. Note that this is $O(n \log n)$ for trees with maximum degree bounded by a constant and $O(n^{1.5})$ for trees with unbounded degrees. For two rooted, *ordered* trees, a maximum agreement subtree can be computed in $O(n \log^2 n)$ time [18]. On the other hand, Amir and Keselman [2] proved that MHT is NP-hard already for three trees with unbounded degrees, but solvable in polynomial time for three or more trees if the degree of at least one of the trees is bounded by a constant. The two fastest known algorithms for MHT with $k \geq 3$ (invented by Bryant [5]

[4] The Dunlin (*Calidris alpina*), a small wading bird breeding in Siberia along the Arctic Sea coastline, is a good example where the geographical distribution drives the evolutionary development. Due to different migrational patterns and differences in climate and food sources, Dunlins in Eastern Siberia and Western Siberia are subject to different selection pressures. Therefore, the species is likely to divide into two separate species, one adapted to wintering in the North Sea area and one in the Pacific. Given enough time, it seems plausible that this process of speciation will continue and lead to a whole range of subspecies distributed along the Arctic coast (see, e.g., [20]).

[5] More precisely, given a set $\mathcal{T}$ of *rooted triplets* (rooted, distinctly leaf-labeled trees containing exactly three leaves each), the problem is to construct a distinctly leaf-labeled tree which contains as many of the rooted triplets in $\mathcal{T}$ as possible as embedded subtrees. This problem is NP-hard for unordered trees [5], [12] but solvable in polynomial time for ordered trees [9]. See also [10] for two polynomial-time approximation algorithms for the unordered case.

[6] UMHT is defined like MHT except that all trees are unrooted and $T \mid S'$ now denotes the tree obtained by first deleting from T all nodes (and their incident edges) not on any path between two leaves in S', and then contracting every node with degree 2.

and Farach et al. [6], respectively) both run in $O(kn^3 + n^d)$ time, where d is an upper bound on at least one of the input trees' degrees; recently, Lee et al. [15] gave an even faster implementation of the algorithm of [5] for the special case $d = 2$.

Hein et al. [11] proved that MHT with three trees with unbounded degrees cannot be approximated within a factor of $2^{\log^{\delta} n}$ in polynomial time for any constant $\delta < 1$, unless NP $\subseteq$ DTIME$[2^{\text{polylog}\, n}]$. This inapproximability result also holds for UMHT [11]. Bonizzoni et al. [4] showed that it can be carried over to MIT restricted to three trees with unbounded degrees as well, and that even stronger bounds can be proved for MIT in the general case. Gąsieniec et al. [10] proved that MHT is hard to approximate in polynomial time even for instances containing only trees of height 2, but showed that if the number of trees is bounded by a constant and all of the input trees' heights are bounded by a constant, then MHT can be approximated within a constant factor in $O(n \log n)$ time.

1.4. *Our Results and Organization of the Paper.* We present the first known polynomial-time algorithms for the uniformly and non-uniformly ordered homeomorphic variants (UOMHT and OMHT) as well as the uniformly and non-uniformly ordered isomorphic variants (UOMIT and OMIT) of the MAST problem. They run in time $O(kn^3)$, $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$, $O(kn^3)$, and $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$, respectively. Our results were obtained by exploiting certain fundamental structural properties of ordered agreement subtrees which allowed for a significant pruning of the otherwise unfeasible number of combinations of subproblems needed for the exact solution.

In Section 2 we introduce some common notation for our algorithms. In Section 3 we present the algorithm for UOMHT. Section 4 is devoted to the algorithm for OMHT. Section 5 describes the algorithms for UOMIT and OMIT.

2. Notation.

We find it convenient to write $S = \{a_1, a_2, \ldots, a_n\}$, where the sequence $a_1, a_2, \ldots, a_n$ is the leaf ordering of T_1.

We also use the following notation. For any $a_l, a_r \in S$ with $l \leq r$, denote by $UOMHT_{a_l,a_r}$ the problem UOMHT under the additional constraint that for any valid solution S', the leaf ordering of $T_1 \mid S'$ must begin with a_l and end with a_r. Furthermore, let $UOMHT_{(a_l),(a_r)}$ be UOMHT restricted to the leaves $\{a_l, a_{l+1}, \ldots, a_r\}$. Observe that while a_l and a_r are required to belong to any solution to $UOMHT_{a_l,a_r}$, they are not necessarily included in a solution to $UOMHT_{(a_l),(a_r)}$. Define $OMHT_{a_l,a_r}$ and $OMHT_{(a_l),(a_r)}$ analogously. Finally, we often write $UOMHT_{a_l,a_r}$, etc. to refer also to an optimal *solution* to the corresponding problem.

3. A Polynomial-Time Algorithm for UOMHT.

In this section we present an algorithm called *All-Pairs* for solving the uniformly ordered maximum homeomorphic agreement subtree problem in $O(kn^3)$ time. Algorithm *All-Pairs* employs dynamic programming to build successively a table of solutions for $UOMHT_{(a_l),(a_r)}$ for all pairs of leaves a_l and a_r with $l \leq r$, using a procedure named *One-Pair* to solve the subproblem $UOMHT_{a_l,a_r}$. Intuitively, to solve $UOMHT_{a_l,a_r}$ (the main computational challenge here), we represent various subsets of S as weighted vertices in "conflict graphs" which contain edges between vertices corresponding to subsets that are not allowed together in

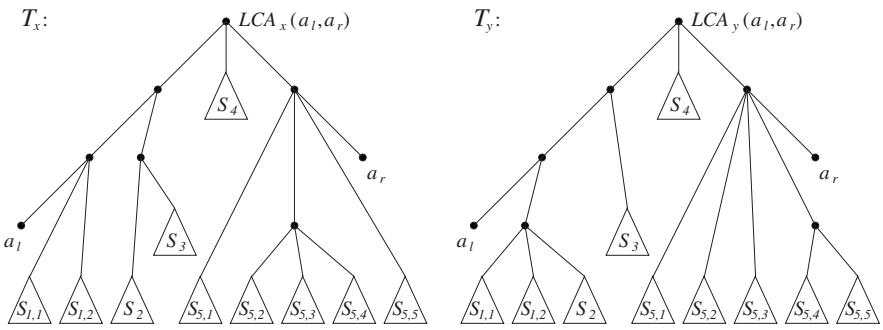

Fig. 1. Two trees T_x and T_y and an induced anchor partitioning $\{S_1, S_2, S_3, S_4, S_5\}$ of the intermediate leaves $\{a_{l+1}, \ldots, a_{r-1}\}$. S_1 and S_5 have been further divided into anchor edge partitionings $\{S_{1,1},\ S_{1,2}\}$ and $\{S_{5,1},\ S_{5,2},\ S_{5,3},\ S_{5,4},\ S_{5,5}\}$, respectively. Note that the anchor for S_2 is also the anchor for S_3 in T_x but not in T_y. Similarly, the anchor edge for $S_{5,4}$ is also the anchor edge for $S_{5,5}$ in T_y but not in T_x.

an agreement subtree, and then look for maximum weighted independent sets in these graphs. Fortunately, due to the leaves being ordered, the constructed conflict graphs have a special structure that allows us to compute maximum weighted independent sets efficiently.

3.1. *Preliminaries*. Before describing the details of our algorithm, we need the following definitions. Please refer to Figure 1 for an illustration.

Suppose that $a_l \in S$ and $a_r \in S$ are fixed and $l \leq r$. For any $T_x \in \mathcal{T}$, let $LCA_x(a_l, a_r)$ be the lowest common ancestor in T_x of a_l and a_r, and let $\mathbf{p}_x(a_l, a_r)$ be the unique path in T_x between a_l and a_r. Any $a_i \in S$ is called an *intermediate leaf* if $l + 1 \leq i \leq r - 1$. For any intermediate leaf a_i and $T_x \in \mathcal{T}$, the *anchor of* a_i in T_x is the node on $\mathbf{p}_x(a_l, a_r)$ where the path from a_i to $LCA_x(a_l, a_r)$ first joins $\mathbf{p}_x(a_l, a_r)$. Similarly, the first edge on the path from the anchor of a_i to a_i is the *anchor edge of* a_i in T_x. Note that the anchor of a_i is equal to the lowest common ancestor of a_l and a_i, or the lowest common ancestor of a_r and a_i. Next, observe that in each $T_x \in \mathcal{T}$, the intermediate nodes can be divided into three types depending on whether their anchors are: (1) in the interior of the path from $LCA_x(a_l, a_r)$ to a_l; (2) equal to $LCA_x(a_l, a_r)$ itself; or (3) in the interior of the path from $LCA_x(a_l, a_r)$ to a_r. We say that an intermediate leaf a_i is *valid* if it belongs to the same type for all $T_x \in \mathcal{T}$. We immediately have:

LEMMA 1. *Suppose $a_l, a_r \in S$ with $l \leq r$. Let a_i be an intermediate leaf. If a_i is not valid then a_i does not belong to any homeomorphic agreement subtree of $\mathcal{T}$ which contains both a_l and a_r.*

PROOF. According to the definition, if a_i is not valid then there exist $T_x, T_y \in \mathcal{T}$ such that a_i belongs to different types in T_x and T_y. This implies $T_x \mid \{a_l, a_i, a_r\} \neq T_y \mid \{a_l, a_i, a_r\}$, and thus $T_x \mid S' \neq T_y \mid S'$ for any S' that contains a_l, a_i, a_r. □

Each pair $a_l, a_r \in S$ with $l \leq r$ induces an *anchor partitioning* $\{S_1, \ldots, S_m\}$ of its set of valid intermediate leaves, in which two leaves b, c belong to the same S_p if and only

if it holds in every $T_x \in \mathcal{T}$ that the anchor of b is also the anchor of c. The leaf indices in any such S_p form a consecutive subsequence of the sequence of valid intermediate leaves, and so we number the sets in the anchor partitioning so that for any $a_i \in S_p$ and $a_j \in S_q$, we have $i < j$ if and only if $p < q$. Each set S_p in an anchor partitioning is further divided into an *anchor edge partitioning* $\{S_{p,1}, \ldots, S_{p,m_p}\}$, where two leaves b, c belong to the same $S_{p,s}$ if and only if it holds in every $T_x \in \mathcal{T}$ that the anchor edge of b is also the anchor edge of c. As above, we number the sets so that for any $a_i \in S_{p,s}$ and $a_j \in S_{p,t}$, we have $i < j$ if and only if $s < t$. Anchor partitionings and anchor edge partitionings have the following useful structural properties:

LEMMA 2. *Suppose $a_l, a_r \in S$ with $l \leq r$. Let $\{S_1, \ldots, S_m\}$ be an anchor partitioning of the valid intermediate leaves, and let A be a homeomorphic agreement subtree of $\mathcal{T}$ which contains a_l and a_r. For any $b \in S_p$ and $c \in S_q$ with $p \neq q$, both b and c can belong to A if and only if the anchor of b is different from the anchor of c in all $T_x \in \mathcal{T}$.*

PROOF. First, suppose there exists a $T_x \in \mathcal{T}$ in which b and c have the same anchor. Because b and c belong to different sets in the anchor partitioning, there always exists a $T_y \in \mathcal{T}$ in which the anchor of b is different from the anchor of c. Then $T_x \mid \{a_l, b, c, a_r\} \neq T_y \mid \{a_l, b, c, a_r\}$, and thus $T_x \mid S' \neq T_y \mid S'$ for any S' that contains a_l, b, c, a_r. Hence, both b and c cannot belong to A in this case.

On the other hand, suppose the anchor of b is different from the anchor of c in every $T_x \in \mathcal{T}$. Since b precedes c in all input trees' leaf orderings (or vice versa) and since b and c are valid (i.e., the anchors of b lie on the same section of $\mathbf{p}_x(a_l, a_r)$ for all $T_x \in \mathcal{T}$ and similarly for c), we have $T_x \mid \{a_l, b, c, a_r\} = T_y \mid \{a_l, b, c, a_r\}$ for all $T_x, T_y \in \mathcal{T}$. Hence, both b and c may belong to A in this case. □

LEMMA 3. *Suppose $a_l, a_r \in S$ with $l \leq r$. Let $\{S_{p,1}, \ldots, S_{p,m_p}\}$ be an anchor edge partitioning of a set S_p in an anchor partitioning, and let A be a homeomorphic agreement subtree of $\mathcal{T}$ which contains a_l and a_r. For any $b \in S_{p,s}$ and $c \in S_{p,t}$ with $s \neq t$, both b and c can belong to A if and only if the anchor edge of b is different from the anchor edge of c in all $T_x \in \mathcal{T}$.*

PROOF. Analogous to the proof of Lemma 2. □

3.2. *Description of the Algorithm.* Algorithm *All-Pairs* and its main procedure *One-Pair* are listed in Figures 2 and 3, respectively. They work as follows.

All-Pairs uses straightforward dynamic programming to compute and store $UOMHT_{(a_l),(a_r)}$ for all a_l, a_r with $l \leq r$ in a table. To obtain $UOMHT_{(a_l),(a_r)}$, *All-Pairs* takes the largest of the previously computed optimal solutions to the two subproblems $UOMHT_{(a_l),(a_{r-1})}$ and $UOMHT_{(a_{l+1}),(a_r)}$ (both stored in the dynamic programming table) and $UOMHT_{a_l,a_r}$, computed by the procedure *One-Pair* as explained below. Finally, *All-Pairs* returns $UOMHT_{(a_1),(a_n)}$.

Given indices l and r, *One-Pair* computes $UOMHT_{a_l,a_r}$ by first identifying the set of valid intermediate leaves in Step 1. According to Lemma 1, the non-valid intermediate leaves cannot be part of a solution and are therefore removed from further considera-

Algorithm *All-Pairs*
Input: An instance of UOMHT.
Output: The subset of leaves in a maximum agreement subtree of $\mathcal{T}$.
  for $length = 1$ to n **do**
    for $l = 1$ to $n - length + 1$ **do**
      $r := l + length - 1$
      Take the largest of $UOMHT_{(a_l),(a_{r-1})}$, $UOMHT_{(a_{l+1}),(a_r)}$, and *One-Pair*(l, r), and store it for entry $UOMHT_{(a_l),(a_r)}$ in the table.
    endfor
  endfor
  return $UOMHT_{(a_1),(a_n)}$
End *All-Pairs*

Fig. 2. The dynamic programming algorithm *All-Pairs* for solving UOMHT.

tion. Step 2 constructs an anchor partitioning $\{S_1, \ldots, S_m\}$ of the remaining (i.e., valid) intermediate leaves, and Step 3 constructs an anchor edge partitioning $\{S_{p,1}, \ldots, S_{p,m_p}\}$ for each set S_p in the anchor partitioning. In Step 4 we define a set $L_{p,s}$ for each $S_{p,s}$ as $UOMHT_{(a_i),(a_j)}$ (obtained directly from the dynamic programming table), where i and j are the smallest and largest indices such that $a_i, a_j \in S_{p,s}$. Then, in Step 5, for each S_p, the algorithm builds a vertex-weighted undirected graph $G_p = (V_p, E_p)$ called a *conflict graph*, defined as follows: Let $V_p = \{S_{p,1}, \ldots, S_{p,m_p}\}$ and set the weight of each

Algorithm *One-Pair*
Input: An instance of UOMHT and the indices of two leaves $a_l, a_r \in S$ with $l \le r$.
Output: $UOMHT_{a_l,a_r}$.
1 Identify the set of valid intermediate leaves.
2 Construct an anchor partitioning $\{S_1, \ldots, S_m\}$ of the valid intermediate leaves.
  for $p = 1$ to m **do**
3 Construct an anchor edge partitioning $\{S_{p,1}, \ldots, S_{p,m_p}\}$ for S_p.
4 For each $S_{p,s}$ in the anchor edge partitioning, let $L_{p,s} = UOMHT_{(a_i),(a_j)}$, where i and j are the smallest and largest indices satisfying $a_i, a_j \in S_{p,s}$.
5 Build a conflict graph $G_p = (V_p, E_p)$ with $V_p = \{S_{p,1}, \ldots, S_{p,m_p}\}$ using the weights $|L_{p,1}|, \ldots, |L_{p,m_p}|$. Compute a maximum weighted independent set M_p in G_p and let L_p be the union of all sets $L_{p,s}$ for which $S_{p,s} \in M_p$.
  endfor
6 Build a conflict graph $G' = (V', E')$ with $V' = \{S_1, \ldots, S_m\}$ using the weights $|L_1|, \ldots, |L_m|$. Compute a maximum weighted independent set M' in G'.
7 **return** the union of $\{a_l, a_r\}$ and all sets L_p for which $S_p \in M'$.
End *One-Pair*

Fig. 3. The procedure *One-Pair* for computing $UOMHT_{a_l,a_r}$.

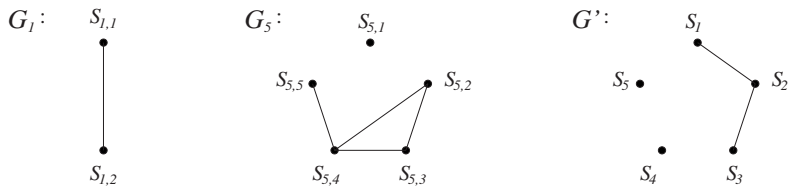

Fig. 4. The conflict graphs G_1 and G_5 for the anchor edge partitionings of S_1 and S_5 in Figure 1 and the conflict graph G' for the anchor partitioning of the intermediate leaves.

$S_{p,s} \in V_p$ to the cardinality of $L_{p,s}$. Furthermore, for all $s \neq t$, let $\{S_{p,s}, S_{p,t}\} \in E_p$ if and only if there exist $b \in S_{p,s}$ and $c \in S_{p,t}$ such that b and c have the same anchor edge in some $T_x \in \mathcal{T}$. See Figure 4 for an example. After building G_p, Step 5 subsequently computes a maximum weighted independent set M_p and constructs the set L_p consisting of all $L_{p,s}$ for which $S_{p,s} \in M_p$. Next, in Step 6, *One-Pair* similarly builds a conflict graph $G' = (V', E')$ where $V' = \{S_1, \ldots, S_m\}$, the weight of each $S_p \in V'$ equals the cardinality of L_p, and for all $p \neq q$, it holds that $\{S_p, S_q\} \in E'$ if and only if there exist $b \in S_p$ and $c \in S_q$ such that b and c have the same anchor in some $T_x \in \mathcal{T}$. Finally, in Step 7, *One-Pair* returns the union of all sets L_p that correspond to vertices in a maximum weighted independent set of G' together with a_l and a_r. Note that if there are no valid intermediate leaves (for example in the case $l = r$), G' has no vertices and *One-Pair* will simply return the set $\{a_l, a_r\}$.

3.3. *Correctness*. We now prove the correctness of the algorithm.

LEMMA 4. *The union L_p of the sets of leaves corresponding to vertices in M_p computed by* One-Pair *in Step* 5 *forms a maximum subset of the leaves in S_p belonging to a homeomorphic agreement subtree which contains a_l and a_r.*

PROOF. Firstly, since M_p is an independent set in G_p, the anchor edge of b differs from the anchor edge of c in every $T_x \in \mathcal{T}$ for every pair $b \in S_{p,s}$ and $c \in S_{p,t}$ with $S_{p,s}, S_{p,t} \in M_p$ and $s \neq t$. By Lemma 3, both b and c (and by repeating this argument, leaves from all $S_{p,s}$-sets in M_p) may belong to a homeomorphic agreement subtree which contains a_l and a_r. For each $S_{p,s}$, the set $L_{p,s}$ gives a subset of the leaves in $S_{p,s}$ that can exist together in such a subtree. Hence, L_p is a valid solution.

Next, let F_p be any subset of S_p that belongs to a homeomorphic agreement subtree A containing a_l and a_r, and let W_p be the set of vertices in G_p induced by F_p, i.e., $S_{p,s} \in W_p$ if and only if some $b \in S_{p,s}$ belongs to F_p. Observe that W_p forms an independent set in G_p (if there exist $S_{p,s}, S_{p,t} \in W_p$ connected by an edge in G_p then there are $b, c \in F_p$ such that $b \in S_{p,s}$, $c \in S_{p,t}$ while b and c have the same anchor edge in some $T_x \in T$; this means that both b and c cannot belong to A according to Lemma 3, which is a contradiction). Now, suppose for the sake of contradiction that $|F_p| > |L_p|$. Since M_p is a maximum weighted independent set with weight $|L_p|$, it follows that the weight of W_p is at most $|L_p|$. Hence, $|F_p|$ is strictly greater than the weight of W_p. Then, for some $S_{p,s}$ in the anchor edge partitioning, F_p contains more than $|L_{p,s}|$ leaves. However, this is a contradiction because every $L_{p,s}$ is guaranteed by its definition to form a maximum subset of the leaves in $S_{p,s}$ that can belong to A. Thus, L_p is optimal. □

LEMMA 5. *The leaves (except a_l and a_r) returned by* One-Pair *in Step* 7 *form a maximum subset of the valid intermediate leaves belonging to a homeomorphic agreement subtree which contains a_l and a_r.*

PROOF. Analogous to the proof of Lemma 4, but using Lemma 2 instead of Lemma 3, and using the maximality of L_p guaranteed by Lemma 4 for every S_p in the anchor partitioning. □

THEOREM 1. *Algorithm* All-Pairs *correctly computes $UOMHT_{(a_1),(a_n)}$.*

PROOF. By Lemma 5, *One-Pair*(l, r) computes each $UOMHT_{a_l,a_r}$ correctly. The theorem now follows from the fact that the size of $UOMHT_{(a_l),(a_r)}$ equals the size of the largest of the three sets $UOMHT_{a_l,a_r}$, $UOMHT_{(a_l),(a_{r-1})}$, and $UOMHT_{(a_{l+1}),(a_r)}$ since either both a_l and a_r belong to the solution or at least one of a_l and a_r does not. □

3.4. *Running Time Analysis.* In this subsection we consider the running time of Algorithm *All-Pairs*.

LEMMA 6. *Each call to* One-Pair *takes $O(kn)$ time.*

PROOF. To implement Steps 1–3, perform the following: For every $T_x \in \mathcal{T}$, construct $\hat{T}_x := T_x \mid \{a_l, a_{l+1}, \ldots, a_r\}$, and augment every internal node v in $\hat{T}_x$ with two integers $A(v)$ and $B(v)$ equal to the index of the leftmost and rightmost leaf, respectively, in the subtree rooted at v. All these indices are obtained by a single postorder traversal of $\hat{T}_x$. Then augment every intermediate leaf with a pointer to its anchor and anchor edge in $\hat{T}_x$ by using the $A(v)$- and $B(v)$-values for the children of each node v on the path between a_l and a_r. (After all trees have been handled, each $a_i \in \{a_{l+1}, \ldots, a_{r-1}\}$ has a total of $2k$ pointers.) Now determine if each intermediate leaf is valid by checking the location of its anchor in all k trees. Next, for each $\hat{T}_x$, process the valid intermediate leaves in left-to-right order and record whenever the anchor or anchor edge changes; the union of the recorded changes will then divide the leaves into the sought sequence of subsets. The above operations take a total of $O(kn)$ time.

Building the conflict graph G_p in Steps 4 and 5 could take $\Omega(n^2)$ time if we list all edges explicitly. Note however that since the leaves are uniformly ordered, if there is a conflict between $S_{p,r}$ and $S_{p,s}$ for some $r \leq s$ then there is also a conflict between $S_{p,t}$ and $S_{p,s}$ for every $r \leq t \leq s$, where we say there is a *conflict* between $S_{p,r}$ and $S_{p,s}$ for $r \neq s$ if there exist $b \in S_{p,s}$ and $c \in S_{p,t}$ such that b and c have the same anchor edge in some $T_x \in \mathcal{T}$, and where $S_{p,s}$ is considered to conflict with itself. We therefore construct and represent the edge set E_p implicitly by storing, for each vertex $S_{p,s} \in V_p$, an integer $R(s)$ defined as the smallest index for which there is a conflict between $S_{p,R(s)}$ and $S_{p,s}$. We can obtain the value of $R(s)$ for any $S_{p,s}$ in $O(k)$ time by taking the minimum of all its k anchor edges' lower nodes' $A(v)$-values and then checking which $S_{p,R(s)}$ that the leaf having that index belongs to. To find a maximum weighted independent set in G_p in Step 5, define $W(0) = 0$ and $W(s)$ for $1 \leq s \leq m_p$ as the maximum weight of an independent set in G_p restricted to the vertices $\{S_{p,1}, \ldots, S_{p,s}\}$, and observe that, for

$1 \leq s \leq m_p$, $W(s)$ is equal to $\max\{W(s-1), |L_{p,s}| + W(R(s)-1)\}$. Hence, we can use dynamic programming to compute $W(m_p)$ in $O(kn)$ time, and then trace back to find a corresponding M_p.

Step 6 is implemented in the same way as Step 5 above. In total, the running time of *One-Pair* is $O(kn)$. □

THEOREM 2. *Algorithm* All-Pairs *runs in* $O(kn^3)$ *time.*

PROOF. Computing one entry $UOMHT_{(a_l),(a_r)}$ in the dynamic programming table takes $O(n)$ time plus the time used by *One-Pair*, which is $O(kn)$ by Lemma 6. There are $O(n^2)$ entries in the table; hence, the total running time is $O(kn^3)$. □

We remark that for even greater efficiency, we can apply a preprocessing step to the input $\mathcal{T}$ before running *All-Pairs* that uses a total of $O(kn)$ time to augment every $s_i \in \{s_1, \ldots, s_{n-1}\}$ with a pointer to the lowest common ancestor in T_x of s_i and s_{i+1} for each $T_x \in \mathcal{T}$. Then we can reconstruct each tree $\hat{T}_x$ in *One-Pair*(l, r) in $O(r - l + 1)$ time, and the running time of *One-Pair*(l, r) reduces to $O(k \cdot (r - l + 1))$. In the preprocessing step we may also compute and store all $A(v)$- and $B(v)$-values. However, these modifications will not improve the asymptotic time complexity of *All-Pairs*.

4. A Polynomial-Time Algorithm for OMHT. We now present an algorithm for the (non-uniformly) OMHT problem running in $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$ time. It has the same overall structure as the algorithm presented in Section 3, but uses other techniques to identify maximum independent sets in the constructed conflict graphs since the simple $O(kn)$-time method described in the proof of Lemma 6 does not work here.

4.1. *Description of the Algorithm.* The main algorithm, named *All-Pairs-OMHT*, is identical to *All-Pairs* in Section 3 except that it computes and stores the successive values of $OMHT_{(a_l),(a_r)}$ and $OMHT_{a_l,a_r}$ instead of $UOMHT_{(a_l),(a_r)}$ in a table. Accordingly, it calls a procedure named *One-Pair-OMHT* to compute each $OMHT_{a_l,a_r}$.

One-Pair-OMHT is listed in Figure 5. Given indices l and r, it proceeds as *One-Pair*, with the following modifications:

First, an additional test is performed in Step 0 to ensure that a_l and a_r appear in the correct order in the leaf orderings of all input trees. If not, the solution to $OMHT_{a_l,a_r}$ is the empty set and the procedure returns $\emptyset$.

Secondly, we extend the definition of an intermediate leaf to the non-uniformly ordered variant of MHT as follows. Suppose that $a_l, a_r \in S$ with $l \leq r$ are fixed as in Section 3.1. We say that any $a_i \in S$ is an *intermediate leaf* if a_i appears after a_l and before a_r in the leaf ordering of every $T_x \in \mathcal{T}$. (The other definitions and lemmas given in Section 3.1 then follow without any further modifications because if a leaf appears before a_l or after a_r in the leaf ordering of some input tree then it cannot belong to $OMHT_{a_l,a_r}$ and may therefore be ignored.) Throughout this section we always use the extended version of the definition.

Thirdly, in Step 4, let $L_{p,s}$ be the largest of all sets $OMHT_{c,d}$ satisfying $c, d \in S_{p,s}$, obtained from the dynamic programming table.

Algorithm *One-Pair-OMHT*

Input: An instance of OMHT and the indices of two leaves $a_l, a_r \in S$ with $l \leq r$.

Output: $OMHT_{a_l,a_r}$.

0 **if** a_l appears after a_r in the leaf ordering of some $T_x \in \mathcal{T}$ **then return** $\emptyset$.

1 Identify the set of valid intermediate leaves.

2 Construct an anchor partitioning $\{S_1, \ldots, S_m\}$ of the valid intermediate leaves.

for $p = 1$ to m **do**

3 Construct an anchor edge partitioning $\{S_{p,1}, \ldots, S_{p,m_p}\}$ for S_p.

4 For each $S_{p,s}$ in the anchor edge partitioning, let $L_{p,s}$ be the largest of all sets $OMHT_{c,d}$ that satisfy $c, d \in S_{p,s}$.

5 Build a conflict graph $G_p = (V_p, E_p)$ with $V_p = \{S_{p,1}, \ldots, S_{p,m_p}\}$ using the weights $|L_{p,1}|, \ldots, |L_{p,m_p}|$ and the points $Q(S_{p,1}), \ldots, Q(S_{p,m_p})$. Compute a maximum weighted independent set M_p in G_p and let L_p be the union of all sets $L_{p,s}$ for which $S_{p,s} \in M_p$.

endfor

6 Build a conflict graph $G' = (V', E')$ with $V' = \{S_1, \ldots, S_m\}$ using the weights $|L_1|, \ldots, |L_m|$ and the points $P(S_1), \ldots, P(S_m)$. Compute a maximum weighted independent set M' in G'.

7 **return** the union of $\{a_l, a_r\}$ and all sets L_p for which $S_p \in M'$.

End *One-Pair-OMHT*

Fig. 5. The procedure *One-Pair-OMHT* for computing $OMHT_{a_l,a_r}$.

Lastly, the edge sets of the conflict graphs constructed in Steps 5 and 6 are modified to include also conflicts between sets in the anchor partitioning (and anchor edge partitionings) arising from them being differently relatively ordered in the input trees, i.e., not just conflicts due to the sharing of an anchor (or anchor edge) in some input tree. For this purpose, to each intermediate leaf b, associate a point $P(b) = (P(b)_1, \ldots, P(b)_k) \in \mathbb{N}^k$, where $P(b)_i$ for $i \in \{1, \ldots, k\}$ equals the number of edges between a_l and the anchor of b in tree T_i. Note that for any two valid intermediate leaves b, c, it holds that b and c belong to the same set S_p in the anchor partitioning if and only if $P(b) = P(c)$. Hence, any set S_p in the anchor partitioning can be represented by a single k-dimensional point $P(S_p) \in \mathbb{N}^k$. In the same way, associate to each intermediate leaf b a point $Q(b) = (Q(b)_1, \ldots, Q(b)_k) \in \mathbb{N}^k$, where $Q(b)_i$ for $i \in \{1, \ldots, k\}$ equals the rank of the anchor edge of b (i.e., its number in the left-to-right ordering of all edges incident to the anchor of b) in tree T_i. As above, it follows that any set $S_{p,s}$ in the anchor edge partitioning of a set S_p can be represented by a single k-dimensional point $Q(S_{p,s}) \in \mathbb{N}^k$. Say that a point $x = (x_1, \ldots, x_k) \in \mathbb{N}^k$ *strictly dominates* a point $y = (y_1, \ldots, y_k) \in \mathbb{N}^k$ if $x_i > y_i$ for all $i \in \{1, \ldots, k\}$, and that if x does not strictly dominate y and y does not strictly dominate x then x and y are *incomparable*. We now define the edge set E_p of any conflict graph G_p by the relation $\{S_{p,s}, S_{p,t}\} \in E_p$ if and only if $s \neq t$ and $Q(S_{p,s})$ and $Q(S_{p,t})$ are incomparable, and the edge set E' of the conflict graph G' by $\{S_p, S_q\} \in E'$ if and only if $p \neq q$ and $P(S_p)$ and $P(S_q)$ are incomparable.

4.2. *Correctness*. To prove the correctness of this approach, we need the next two lemmas which are the counterparts of Lemmas 2 and 3.

LEMMA 7. *Suppose $a_l, a_r \in S$ with $l \leq r$. Let $\{S_1, \ldots, S_m\}$ be an anchor partitioning of the valid intermediate leaves, and let A be a homeomorphic agreement subtree of $\mathcal{T}$ which contains a_l and a_r. For any $b \in S_p$ and $c \in S_q$ with $p \neq q$, both b and c can belong to A if and only if $P(S_p)$ strictly dominates $P(S_q)$ or $P(S_q)$ strictly dominates $P(S_p)$.*

PROOF. First, suppose $P(S_p)$ and $P(S_q)$ are incomparable. Take any $T_x \in \mathcal{T}$ in which the anchor of b is different from the anchor of c (by definition, such a T_x always exists), and assume without loss of generality that $P(S_p)_x < P(S_q)_x$. Because $P(S_p)$ and $P(S_q)$ are incomparable, for some $T_y \in \mathcal{T}$ we have $P(S_p)_y \geq P(S_q)_y$. This means that in T_x, the anchor of b is closer to a_l than the anchor of c is, but that this is not true in T_y. Then $T_x \mid \{a_l, b, c, a_r\} \neq T_y \mid \{a_l, b, c, a_r\}$, and thus $T_x \mid S' \neq T_y \mid S'$ for any S' that contains a_l, b, c, a_r. Hence, both b and c cannot belong to A in this case.

On the other hand, suppose $P(S_q)$ strictly dominates $P(S_p)$ (the case $P(S_p)$ strictly dominates $P(S_q)$ is symmetric). Then, in every $T_x \in \mathcal{T}$ it holds that the anchor of b is closer to a_l than the anchor of c is. Since b and c are valid, $T_x \mid \{a_l, b, c, a_r\} = T_y \mid \{a_l, b, c, a_r\}$ for all $T_x, T_y \in \mathcal{T}$. Hence, both b and c may belong to A in this case. □

LEMMA 8. *Suppose $a_l, a_r \in S$ with $l \leq r$. Let $\{S_{p,1}, \ldots, S_{p,m_p}\}$ be an anchor edge partitioning of a set S_p in an anchor partitioning, and let A be a homeomorphic agreement subtree of $\mathcal{T}$ which contains a_l and a_r. For any $b \in S_{p,s}$ and $c \in S_{p,t}$ with $s \neq t$, both b and c can belong to A if and only if $Q(S_{p,s})$ strictly dominates $Q(S_{p,t})$ or $Q(S_{p,t})$ strictly dominates $Q(S_{p,s})$.*

PROOF. Analogous to the proof of Lemma 7. □

Clearly, the leaves in any set of the form $OMHT_{c,d}$ with $c, d \in S_{p,s}$ can exist together with a_l and a_r in a homeomorphic agreement subtree of $\mathcal{T}$. It follows as in the proof of Lemma 4 (using Lemma 8 instead of Lemma 3) that the union of the $L_{p,s}$-sets corresponding to vertices in a maximum weighted independent set computed in Step 5 indeed yields a maximum subset of the leaves in S_p belonging to a homeomorphic agreement subtree which contains a_l and a_r. As in Lemma 5 (but using Lemma 7), we then see that *One-Pair-OMHT*(l, r) computes each $OMHT_{a_l,a_r}$ correctly.

From the above and the proof of Theorem 1, we obtain:

THEOREM 3. *Algorithm* All-Pairs-OMHT *correctly computes* $OMHT_{(a_1),(a_n)}$.

4.3. *Running Time Analysis*. Here we analyze the time complexity of Algorithm *All-Pairs-OMHT*.

First examine *One-Pair-OMHT*. Step 0 is straightforward. To implement Steps 1–3, let I be a $(k \times n)$-matrix whose entries are initialized to 0, and for each $T_x \in \mathcal{T}$, set $I[x, i]$ to 1 for every a_i that appears after a_l and before a_r in the leaf ordering of T_x. Then determine for each $a_i \in S$ if it is an intermediate leaf by checking if $I[x, i] = 1$

for all $x \in \{1, \ldots, k\}$. Next, compute the $P(a_i)$- and $Q(a_i)$-values as follows: for every $T_x \in \mathcal{T}$, start at a_l and traverse $\mathbf{p}_x(a_l, a_r)$ while keeping track of the distance traveled and filling in the values for $P(a_i)_x$ and $Q(a_i)_x$ for all intermediate leaves a_i belonging to subtrees rooted at the children of the current node. After this, it is easy to identify and remove those intermediate leaves which are not valid. Find the anchor partitioning by lexicographically sorting the $P(a_i)$-values for all valid intermediate leaves (recall that each set in the anchor partitioning corresponds to one distinct point in $\mathbb{N}^k$), and similarly find each anchor edge partitioning by lexicographically sorting the respective $Q(a_i)$-values. The lexicographic sorts take a total of $O(kn)$ time (see, for instance, [1]), so all the above operations can be performed in $O(kn)$ time.

To implement Step 4 efficiently, also store the cardinality of each entry $OMHT_{a_l,a_r}$ in the dynamic programming table; then, since all leaf sets of the form $S_{p,s}$ are disjoint, the sets $L_{p,s}$ can be looked up using a total of $O(n^2)$ time.

When computing maximum weighted independent sets in the conflict graphs in Steps 5 and 6, we can no longer apply the fast method from the proof of Lemma 6 since the leaves are not necessarily uniformly ordered. Below, we present two methods to represent a conflict graph G' and compute a maximum weighted independent set in it efficiently for OMHT (the same methods are also used for the conflict graphs of the form G_p). As we shall see, the two methods' running times depend differently on k.

Method 1: Make use of the fact that G' is a k-trapezoid graph and apply a known fast algorithm [8] to find a maximum weighted independent set in such a graph.

For convenience, we recall the following definition (see, e.g., [8] for more details). An undirected graph $G = (V, E)$ is a *k-trapezoid graph* if there exist two mappings $l : V \to \mathbb{N}^k$ and $u : V \to \mathbb{N}^k$ such that for any $a, b \in V$ it holds that $\{a, b\} \in E$ if and only if $l(a)$ does not strictly dominate $u(b)$ and $l(b)$ does not strictly dominate $u(a)$.[7] Any triple (V, l, u) which satisfies the above condition is called a *k-dimensional box representation of G* and contains enough information to reconstruct the edge set E. Now we can see that the conflict graph G' is a k-trapezoid graph (in fact, a special case of a k-trapezoid graph in which all trapezoids in the geometric interpretation are line segments) by setting $l(S_p) = u(S_p) = P(S_p)$ for all $S_p \in V'$ in the definition above. Then the following result from [8] applies:

LEMMA 9 [8]. *Given a k-dimensional box representation of a k-trapezoid graph G and the weights of the vertices in G, a maximum weighted independent set of G can be found in $O(N \log^{k-1} N)$ time, where N is the number of vertices in G.*

By Lemma 9, Steps 5 and 6 of *One-Pair-OMHT* can be implemented to run in $O(n \log^{k-1} n)$ total time using Method 1.

Method 2: Represent G' as a set $\mathcal{W}$ of strings and find a longest common subsequence of $\mathcal{W}$.

[7] To interpret this condition geometrically, let $H_1, \ldots, H_k$ be k horizontal lines and let A and B be the union of the $k - 1$ trapezoids formed by the points whose coordinates on each H_i are $l(a)_i$ and $r(a)_i$, and $l(b)_i$ and $r(b)_i$, respectively. Then $l(a)$ does not strictly dominate $u(b)$ and $l(b)$ does not strictly dominate $u(a)$ if and only if A and B intersect.

More precisely, encode each $S_p \in V'$ by a symbol σ_p and define an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$, where $m = |V'|$ as before. For each $T_x \in \mathcal{T}$, construct two strings E_x and F_x of length m in which each symbol in Σ occurs exactly once (i.e., E_x and F_x are permutations of Σ) such that for every $p, q \in \{1, \ldots, m\}$ it holds that: (1) if $P(S_p)_x < P(S_q)_x$ then σ_p precedes σ_q in both E_x and F_x; and (2) if $P(S_p)_x = P(S_q)_x$ then σ_p precedes σ_q in exactly one of E_x and F_x. This ensures that any common subsequence of the $2k$ strings in the set $\{E_1, F_1, \ldots, E_k, F_k\}$ cannot contain a pair of symbols σ_p and σ_q for which $P(S_p)$ and $P(S_q)$ are incomparable. Next, account for the weight w of each vertex $S_p \in V'$ by replacing every occurrence of σ_p in the $2k$ strings by a sequence of w new symbols $\sigma_p^1, \ldots, \sigma_p^w$, ordered in the same way in all strings. Let $\mathcal{W}$ be the resulting set of strings. It follows from the above that a longest common subsequence of $\mathcal{W}$ will yield an encoding of a maximum weighted independent set of G'. The length of each string in $\mathcal{W}$ is the sum of all weights of vertices in V', which is always less than n because the vertices in V' correspond to disjoint subsets of S. Therefore, we can use the next lemma to find a maximum weighted independent set of G' in $O(kn^2)$ time.

LEMMA 10. *Let $\mathcal{W}$ be a set of strings over an alphabet Σ such that each symbol from Σ occurs exactly once in every string. Then a longest common subsequence of $\mathcal{W}$ can be computed in $O(|\mathcal{W}| \cdot |\Sigma|^2)$ time.*

PROOF. First compute and store the ranks of all symbols in all strings in a $(|\mathcal{W}| \times |\Sigma|)$-table in $O(|\mathcal{W}| \cdot |\Sigma|)$ time. For any $a, b \in \Sigma$, we can then determine if a occurs before b in all strings, if b occurs before a in all strings, or neither in $O(|\mathcal{W}|)$ time. Build a directed graph $D = (\Sigma, E)$ where $(a, b) \in E$ if and only if a occurs before b in all strings. D is clearly acyclic and can be constructed in $O(|\mathcal{W}| \cdot |\Sigma|^2)$ time. A longest common subsequence of $\mathcal{W}$ corresponds to a longest directed path in D, which can be computed within the given time bound by doing a topological sort of D and then applying simple dynamic programming to the vertices in topological order. □

Hence, Steps 5 and 6 of *One-Pair-OMHT* can be implemented to run in $O(kn^2)$ total time using Method 2.

The running time of Method 1 is better than that of Method 2 for small k, but superpolynomial for large k. By taking the faster of Method 1 and Method 2, the running time for Steps 5 and 6 becomes $O(\min\{kn^2,\ n \log^{k-1} n\})$.

To summarize, each call to *One-Pair-OMHT* takes $O(kn+n^2+\min\{kn^2, n \log^{k-1} n\}) = O(n \min\{kn,\ n + \log^{k-1} n\})$ time. As in Theorem 2, we have:

THEOREM 4. *Algorithm* All-Pairs-OMHT *runs in $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$ time.*

5. Polynomial-Time Algorithms for UOMIT and OMIT. Our polynomial-time algorithms for UOMHT and OMHT can easily be adapted to UOMIT and OMIT, respectively. In order to solve the corresponding subproblems $UOMIT_{a_l,a_r}$ and $OMIT_{a_l,a_r}$, first note that if a_l and a_r belong to an isomorphic subtree of $\mathcal{T}$ then the path between a_l and $LCA_x(a_l, a_r)$ must have the same length in every $T_x \in \mathcal{T}$, and likewise for the path

between a_r and $LCA_x(a_l, a_r)$. Hence, in the first step, if a_l and a_r do not satisfy the above criterion, just return the empty set. Next, identify every valid intermediate leaf a_i for which the distance from a_i to $\mathbf{p}_x(a_l, a_r)$ is equal in all trees $T_x \in \mathcal{T}$ and for which $P(a_i)_1 = \cdots = P(a_i)_k$; if at least one of these two conditions is not satisfied then a_i cannot be part of a solution together with a_l and a_r, and must be discarded. Now there are no conflicts between the sets $S_1, \ldots, S_m$ in the anchor partitioning of the remaining leaves because if two leaves a_i and a_j have the same anchor in some $T_x \in \mathcal{T}$ then a_i and a_j must in fact belong to the same set S_p (to see this, observe that $P(a_i)_x = P(a_j)_x$ plus the above conditions implies $P(a_i) = P(a_j)$ by transitivity). Consequently, the conflict graph G' has no edges and there is no need to use special algorithms for computing maximum weighted independent sets in Step 6 as the optimal solution is trivially V'. However, we may still have conflicts in the graphs G_p in Step 5, so we apply the techniques from Sections 3 and 4 to find maximum weighted independent sets here. As before, we obtain:

THEOREM 5. *UOMIT can be solved in $O(kn^3)$ time.*

THEOREM 6. *OMIT can be solved in $O(n^3 \min\{kn,\ n + \log^{k-1} n\})$ time.*

6. Final Remarks. We have shown that two main variants of the MAST problem which are NP-hard (and NP-hard even to approximate) for unordered trees can in fact be solved efficiently for *ordered* trees. It would be interesting to investigate further other combinatorial problems known to be NP-hard for unordered trees to see if they admit polynomial-time solutions in the ordered case. (Other combinatorial problems where this phenomenon occurs have been studied in, e.g., [9] and [13].)

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[2] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms. *SIAM Journal on Computing*, 26(6):1656–1669, 1997.

[3] H. Bodlaender, R. Downey, M. Fellows, and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147:31–54, 1995.

[4] P. Bonizzoni, G. Della Vedova, and G. Mauri. Approximating the maximum isomorphic agreement subtree is hard. *International Journal of Foundations of Computer Science*, 11(4):579–590, 2000.

[5] D. Bryant. Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis. Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997.

[6] M. Farach, T. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55:297–301, 1995.

[7] M. Fellows, M. Hallett, and U. Stege. Analogs & duals of the MAST problem for sequences & trees. *Journal of Algorithms*, 49(1):192–216, 2003.

[8] S. Felsner, R. Müller, and L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74(1):13–32, 1997.

[9] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. Inferring ordered trees from local constraints. In *Proceedings of Computing*: *the 4th Australasian Theory Symposium* (*CATS* '98), pages 67–76. Volume 20(3) of Australian Computer Science Communications. Springer-Verlag, Singapore, 1998.

[10] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3:183–197, 1999.

[11] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.

[12] J. Jansson. On the complexity of inferring rooted evolutionary trees. In *Proceedings of the Brazilian Symposium on Graphs*, *Algorithms*, *and Combinatorics* (*GRACO* 2001), pages 121–125. Volume 7 of Electronic Notes in Discrete Mathematics. Elsevier, Amsterdam, 2001.

[13] T. Jiang, L. Wang, and K. Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[14] M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001.

[15] C.-M. Lee, L.-J. Hung, M.-S. Chang, C.-B. Shen, and C.-Y. Tang. An improved algorithm for the maximum agreement subtree problem. *Information Processing Letters*, 94(5):211–216, 2005.

[16] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

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

[18] W.-K. Sung. Fast Labeled Tree Comparison via Better Matching Algorithms. Ph.D. thesis, University of Hong Kong, 1998.

[19] V. G. Timkovsky. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25:1–13, 1990.

[20] P. W. Wenink. Mitochondrial DNA sequence evolution in shorebird populations. Dissertation, Wageningen University, 1994.