Inferring Ordered Trees from Local Constraints

Leszek Gąsieniec[1], Jesper Jansson[2], Andrzej Lingas[2], and Anna Östlin[2]

[1] Department of Computer Science, University of Liverpool, Chadwick Building, Peach Street, Liverpool, L69 7ZF, UK, email:leszek@csc.liv.ac.uk

[2] Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden, email: {Jesper.Jansson, Andrzej.Lingas, Anna.Ostlin}@dna.lth.se

Abstract. We consider a problem of inferring an ordered tree from a set of local constraints on its leaves which we term as *the ordered local consensus tree* problem. Using our efficient decremental interval union algorithm, we show that the ordered local consensus tree problem for m constraints on n leaves can be deterministically solved in time $O((m+n)\log n)$. We also show that the related optimization problem of constructing an ordered local consensus tree for the maximum number of 3-leaf constraints is solvable in cubic time.

1 Introduction

1.1 Background

Evolutionary trees are used to model tree-like evolution for species from some given set. The basic idea is to let the leaves correspond to species and to let the internal nodes represent the species' ancestors. Usually, all children in the tree are unordered, i.e., there is no distinction between left and right, since the intrinsic branching structure of the tree is what matters. The problem of constructing a reliable evolutionary tree, given some data, has been studied for a long time [2, 3, 4, 6, 9, 16, 17]. There are different approaches, depending among other things on what kind of data that is available. Also, depending on the nature of the underlying application, the resulting trees may be rooted or unrooted, and with or without weighted edges. Thus, various versions of this problem arise in, for example, computational biology when one wants to find out how different biological species or molecules are related, and comparative linguistics, where it is central to find out how different languages have evolved. In this paper, we focus on a special case of the so-called local consensus tree problem for rooted, unweighted evolutionary trees in which a specified left-right ordering is imposed upon the species. We call this problem *the ordered local consensus tree problem*. The motivations for the ordered versions of the consensus tree problem come from tree drawing and from the desire to somehow order the leaves for ease of presentation, for example by the age of the species or languages involved, or just alphabetically.

1.2 Problem Description

For a set of binary rooted trees $\{T_1, ..., T_m\}$, each one leaf-labeled by a subset $L(T_i)$ of $\{1, 2, ..., n\}$ (the "species"), *the consensus tree problem* asks if there is a tree T such that for $i = 1, ..., m$, T_i is homeomorphic to the subtree of T induced by the nodes in $L(T_i)$ and their ancestors, and if so, what it looks like. In other words, is there a consensus tree that contains all the structural information of $T_1, T_2, ..., T_m$ combined, or do some of the given trees contradict each other? If each of the input trees is known to be of size bounded by a constant, the problem is termed *the local consensus tree problem.* See Figure 1 for an example.

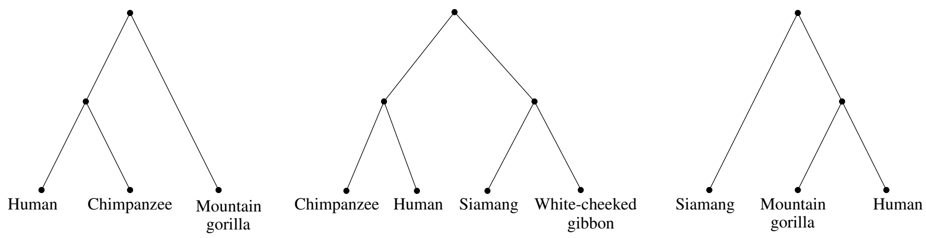

Fig. 1. Are these evolutionary trees compatible? (The answer is yes!)

The problems of efficiently computing consensus trees have been intensively studied recently [10, 11, 15]. Already in the early eighties, Aho, Sagiv, Szymansky, and Ullman provided a quadratic-time solution for a variant of the local consensus tree problem [1]. Using new dynamic data structures for graph connectivity, Henzinger, King, and Warnow presented an $O(m\sqrt{n})$-time deterministic implementation and an $O(m \log^3 n)$ expected-time randomized implementation of the method of Aho *et al.* in [10].

Each one of the input trees can be regarded as a set of *constraints* on the possible phylogenetic relationships between species. A constraint of the form $(\{i, j\}, k)$ is called a *3-leaf constraint* or a *rooted triple*, and indicates that the lowest common ancestor of the leaves corresponding to species i and j is a proper descendant of the lowest common ancestor of the leaves corresponding to species i and k. For example, the leftmost tree in Figure 1 is equivalent to the constraint ({Human, Chimpanzee}, Mountain gorilla). Since any binary rooted tree on n leaves can be defined by a set of $(n - 2)$ 3-leaf constraints [16], we consider the problem of inferring a local consensus tree from a set of 3-leaf constraints rather than from a set of constant-sized trees.

Scanning the labels of the leaves of an ordered tree from left to right yields its *frontier*. If T, in addition to being in compliance with the given constraints, is required to be constructed in such a way that its frontier equals a certain ordering of the species, then we term the resulting problems *the ordered consensus* *tree problem* and *the ordered local consensus tree problem*, respectively. Figure 2 shows an ordered consensus tree for the species in Figure 1.

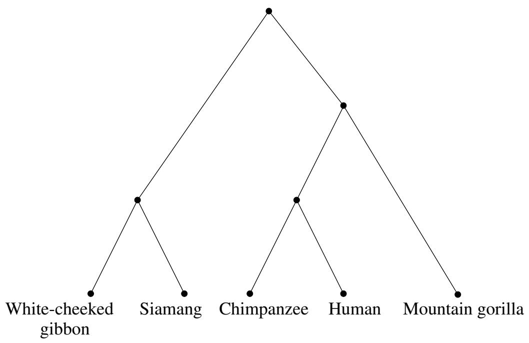

Fig. 2. An ordered consensus tree for the trees depicted in Figure 1. Here, the species are ordered by size.

1.3 Our Results

Our key observation is that the basic dynamic graph used by Henzinger *et al.* in their fast algorithms for the local consensus tree problem becomes an *interval graph* if the tree under construction is required to be ordered. For interval graphs with known interval representation, efficient dynamic connectivity algorithms are simply implied by the known efficient dynamic interval union algorithms [5]. As we need only a decremental interval union algorithm, and the known efficient fully dynamic interval union algorithms are rather complicated, we provide a very simple and practical, decremental interval union algorithm matching the time performance of that from [5] in the decremental case.

Using our decremental interval union algorithm, we show that the ordered local consensus tree problem can be deterministically solved in time $O((m + n)\log n)$. We also show that the related optimization problem of constructing an ordered local consensus tree for the maximum number of given 3-leaf binary trees is solvable in cubic time.

1.4 Overview

The rest of the paper is organized as follows. In Section 2 we recall the definitions of and some known facts about segment and interval trees. In Section 3 we describe our simple decremental interval union algorithm. In Section 4 we present our fast algorithm for the construction of ordered local consensus trees, and in Section 5 we provide the cubic-time solution to the related optimization problem mentioned above.

2 Preliminaries

Segment trees, interval trees and interval tries will be used in our algorithms.

The following facts about segment trees and interval trees are shown in [13].

A segment tree is a useful data structure for storing a set S of intervals on the real line R together with some additional information. The leaves in a segment tree are $(-\infty, x_1)$, $[x_1, x_1]$, (x_1, x_2), $[x_2, x_2]$, ..., (x_N, ∞), where x_i, $i = 1, 2, ..., N$, are the legal endpoints for the segments and $x_i < x_{i+1}$ for all $i = 1, ..., N-1$.

Each node u of a segment tree is augmented by a node list $NL(u)$. To describe NL we will use $xrange(u)$, defined as follows. First, $xrange(root) = (-\infty, +\infty]$. Let r be the right child and l the left child of node u. The split value at node u is any value that is at least as large as the values at the leaves in the left subtree of u and smaller than the values at the leaves in the right subtree of u. Next, we define $xrange(r) = xrange(u) \cap (split(u), +\infty]$ and $xrange(l) = xrange(u) \cap (-\infty, split(u)]$. We set $NL(u)$ to $\{I \in S \mid xrange(u) \subseteq I \ \& \ xrange(parent(u)) \not\subseteq I\}$. Each interval can be stored in at most in $O(\log N)$ nodes.

Fact 1. [13] *A segment tree for a set of n intervals with both endpoints in a universe $\mathcal{U} \subset \mathcal{R}$ of size N can be constructed in time $O(n \log N)$. The depth of the tree is $O(\log N)$ and its size is $O(N)$. An interval with both endpoints in $\mathcal{U}$ can be inserted or deleted from the segment tree in time $O(g(n) \log N)$, where $g(n)$ is the time required to insert or delete an interval from a node list of size n.*

In our application $g(n) = O(1)$, since we shall use a node counter instead of a node list. The value $NC(u)$ of the node counter for a node u is just the cardinality of $NL(u)$, i.e., deletion or insertion corresponds to decreasing or increasing $NC(u)$ by one.

An interval tree is a leaf-oriented binary search tree that supports intersection queries for a set S of closed intervals on the real line in logarithmic time.

Fact 2. [13] *Suppose that the left endpoints of the intervals in S belong to a subset of R of size N and $|S| = n$. An interval tree T of depth $O(\log N)$ for S can be constructed in time $O(N + n \log Nn)$ using space $O(n + N)$. The insertion or deletion of an interval with left endpoint in $\mathcal{U}$ into T takes time $O(\log n + \log N)$. The intersection query is supported by T in time $O(\log N + r)$, where r is the number of the reported intervals.*

The interval trie is another data structure for maintaining a set S of intervals, efficiently supporting the stabbing query. The answer to it is the list of all intervals that contain the query point. The next fact following from [14] characterizes interval tries.

Fact 3. [14] *Let S be a set of n intervals with endpoints in $\{1, ..., N\}$ and let f be a function mapping natural numbers into natural numbers greater than 1. The interval trie data structure performs insertions and deletions of intervals in S in time $O(f(N) + \log\log N)$. The stabbing query can be answered in time $O(\frac{\log N}{\log f(N)} + r)$ where r is the number of reported component intervals. The total space required is $O(N \frac{\log N}{\log f(N)} + nf(N))$.*

3 A Simple Decremental Interval Union Algorithm

Here we consider the problem of maintaining the union of the set S of intervals under a sequence $s_1, ..., s_m$ of interval deletions from S, where for $i = 1, ..., m$, s_i is currently in S.

The first crucial step of our method consists in the construction of a special balanced segment tree T for S [13]. Let $e_1, ..., e_l$ be the consecutive endpoints of the intervals in S. The consecutive leaves in T are $(-\infty, e_1)$, $[e_1, e_1]$, (e_1, e_2), $[e_2, e_2]$, ..., (e_l, ∞). To each node u of T we associate the counter $NC(u)$ instead of the standard node list $NL(u)$. Recall that the value of the counter is just the cardinality of $NL(u)$. Note that $l \leq 2n$. Hence, the segment tree T can be constructed in time $O(n \log n)$ by Fact 1.

Additionally, for each node u of T, we set a special bit $r(u)$ to 1 iff the NC counters along the path from u to the root of T, including u and the root of T, are set to zero. Clearly, we can determine $r(u)$ for all nodes of T by traversing T in a top-down fashion in time proportional to the size of T, i.e., $O(n)$.

Remark 1. *There exist leaves l_1, l_2 and l_3 in the left-right order in T such that $r(l_1) = 0$, $r(l_2) = 1$ and $r(l_3) = 0$ if and only if the union of S is disconnected.*

Finally, we form a linear array A, to store the consecutive leaves of T, in linear time.

Besides the segment tree T, we initialize an interval tree P in order to maintain the component intervals in the union of the current S. We can find the component intervals of the initial S, e.g., by using a standard sweep-line technique in time $O(n \log n)$. Next, we can construct P on $O(n)$ leaves determined by the endpoints of the intervals in S and insert the initial component intervals in time $O(n \log n)$ by Fact 2.

To find the component intervals of the set resulting from the deletion of the interval s_1 from S we proceed as follows.

To begin with, we find the component interval ci_1 that s_1 belongs to in time $O(\log n)$ by querying P with s_1.

Next, we find the $O(\log n)$ nodes u of T whose node lists would contain s_1, if we were keeping them, and decrease $NC(u)$ by one. This can be done in time $O(\log n)$ by Fact 1. If $NC(u)$ drops to 0 and $r(parent(u)) = 1$ then we set $r(u)$ to 1 and update $r(w)$ for the descendants w of u in a top-down and left-right fashion. We also produce the list $LL(u)$ of consecutive leaf-descendants w of u for which $r(w)$ become 1 due to the updating ($LL(u)$ can be empty). Further, we concatenate the lists $LL(u)$ in the order of the ranges of nodes u. Let L_1 denote the resulting list. Note that the updating of $r(w)$'s as well as the construction of the list L_1 takes time proportional to the number r_1 of nodes w that got $r(w)$ set to 1 for the first time.

Now the following observation is crucial: for any pair of consecutive leaves a, b in L_1 that are not neighbors in the array A of consecutive leaves of T, the right neighbor $rn(a)$ of a in A and the left neighbor $ln(b)$ of b in A yield the new component interval. Also, if the left endpoint of the old component interval ci_1 that s_1 belonged to is not on L_1 then it together with the left neighbor $ln(a)$

of the first leaf on L_1 yield another new component interval. Analogously, the right neighbor $rn(z)$ of the last leaf on L_1 together with the right endpoint of ic_1 might yield yet another new component interval. Hence, we need to insert the new component intervals into P and delete ci_1 from P. Any of the aforementioned insertions and deletions takes $O(\log n)$ time.

We conclude that we can update T and P in order to maintain the union of the set of intervals resulting from the deletion of s_1 from S in time $O((k+1)\log n + r_1)$, where k is the number of the new component intervals.

The next deletions d_i, $i = 2, ..., m$, are handled analogously. We define r_i for $i = 2, ..., m$, analogously as r_1. It follows from the definition of r_i's that the sum $\sum_{i=1,...,m} r_i$ is bounded from above by the size of T, i.e., $O(n)$.

Summarizing, we obtain our main result in this section.

Theorem 1. *Let S be a set of n closed intervals on the real line. After $O(n \log n)$-time preprocessing the union of S can be maintained under a sequence of m interval deletions in time $O(m \log n + n + k \log n)$, where k is the final number of component intervals. After each deletion the newly created component intervals can be listed without increasing the asymptotic complexity. The intersection query can be answered in time $O(\log n + r)$ where r is the number of component intervals to report.*

If we need only a stabbing query and the endpoints of the intervals belong to $\{1, ..., N\}$ for a small N, we can improve our result using an interval trie instead of the interval tree P. By Fact 3, we obtain the following variant of the result.

Theorem 2. *Let S be a set of n closed intervals with endpoints in $\{1, ..., N\}$ and let $f(N)$ be an integer greater than 1. After $O(n \log n + N \frac{\log N}{\log f(N)} + nf(N))$-time preprocessing the union of S can be maintained under a sequence of m interval deletions in time $O(m \log n + n + k(f(N) + \log \log N))$, where k is the final number of component intervals. After each deletion the newly created component intervals can be listed without increasing the asymptotic complexity. The stabbing query can be answered in time $O(\frac{\log N}{\log f(N)} + r)$ where r is the number of reported intervals. The total space required is $O(N \frac{\log N}{\log f(N)} + nf(N))$.*

Our simple algorithm for maintaining the union of a set S of intervals under a sequence of interval deletions can also be interpreted (and will be used in the next section) as an algorithm for the connected components of the interval graph induced by S under a sequence of vertex deletions.

If we have an interval graph G and don't have its interval representation S we can form it in time proportional to the size of G, e.g., by ordering the maximal cliques of G so that for each vertex v of G the maximal cliques containing v are consecutive [8, 12].

4 Efficient Construction of an Ordered Consensus Tree

A binary rooted tree T_i on $O(1)$ leaves in the local consensus tree problem corresponds to $O(1)$ 3-leaf constraints of the form $(\{i, j\}, k)$ on the consensus tree T, where $(\{i,j\},k)$ implies that the lowest common ancestor of the leaves i and j has to be a proper descendant of the lowest common ancestor of the leaves i and k [16]. For this reason, we can consider the problem of inferring a local consensus tree from a set of 3-leaf constraints instead of the local consensus tree problem.

Using recent dynamic data structures for graph connectivity, Henzinger *et al.* provide efficient algorithms for the aforementioned problem of inferring a local consensus tree. Their deterministic algorithm takes $O(m\sqrt{n})$ time whereas their randomized algorithm takes $O(m\log^3 n)$ expected-time, where m is the number of 3-leaf constraints on leaf-labels in $\{1,\ 2, ..., n\}$. In both cases, they use the undirected graph U and the directed graph D defined as follows.

- $U = (V,E)$ with V equal to the set $\{1,2,...,n\}$ of leaves and where, for each constraint $(\{a,b\},c)$ in C, edges $\{a,b\}$ and $\{b,c\}$ are in E.
- $D = (V',A)$ where, for each constraint $(\{a,b\},c)$ in C, nodes $\{a,b\}$ and $\{b,c\}$ are in V' and $\{a,b\} \rightarrow \{b,c\}$ is in A.

The graph U is used for finding current (yellow in [10]) connected components. The consensus tree returned is found by combining the trees constructed for the current connected components. The graph D is used for finding edges in U that can be removed (colored red in [10]), these edges correspond to the so-called maximal nodes in D. A *maximal node* in D is a node with no outgoing edges.

Remark 2: *Whenever two leaves a and b are in the same current component of U then not only a and b, but also all other leaves in $[a,b]$ have to be in the same subtree of the ordered consensus tree representing this component.*

By Remark 2, the vertices of U can be interpreted as segments inducing an interval graph. Hence, instead of computing the current components of U we can compute the corresponding components of the interval graph utilizing our efficient decremental interval union algorithm.

By combining this idea with that of the algorithm of Henzinger *et al.*, we can build an ordered consensus tree as follows.

Algorithm 1

1. $S \leftarrow \{[a,b] | (a,b) \in E\}$.
2. Construct D and initialize the interval trie Q on the universe $\{1,2,...,n\}$.
3. Identify maximal vertices in D. Move the corresponding intervals from S to Q.
4. If S is empty, then return the consensus tree O with a root and all nodes in U children of the root. Otherwise, compute the new component intervals of the union of S by using the decremental algorithm of Section 3. If there is no new component interval then return "no ordered consensus tree" and stop. Let $c_1, c_2, ..., c_k$ be the new component intervals in the order of their endpoints. For $i = 1, ..., k-1$, create the root of the ordered consensus tree to be computed for the component interval c_i and link it with that of the

previous interval component including c_i if such one exists. For $i = 1, ..., k-1$, if c_i and c_{i+1} are included in the same old component interval then query Q with the median m_i of the right endpoint of c_i and the left endpoint c_{i+1}. Remove all reported (i.e., containing m_i) intervals from Q and the corresponding vertices from D. Go to Step 2.

Theorem 3. *For a set of m 3-leaf constraints on leaf-labels in $\{1, 2, ..., n\}$, we can decide whether there exists an ordered consensus tree, and if so, construct it, in time $O((m+n)\log n)$.*

Proof. (sketch). If Algorithm 1 produces an ordered consensus tree then it is equivalent to the minimal tree produced by Aho *et al.* algorithm or the tree produced by the faster algorithm of Henzinger *et al.* (Algorithm 1 in [10]). To see this note that the current component intervals of S correspond to the yellow components of the graph U in [10] and the intervals stored in Q correspond to the current red edges of U. Furthermore, an interval stored in Q corresponds to a red separable edge in [10] iff it overlaps with at least two new component intervals, i.e., if it is intersected by at least one of the medians m_i, $i = 1, ..., k-1$. If Algorithm 1 fails to produce an ordered consensus tree then there is no such a tree by Remark 1 and the correctness of the algorithms of Henzinger *et al.*

Analogously as in the algorithm of Henzinger *et al.*, the construction of the graph D takes time $O(m)$ and the overall time taken by determining the maximal vertices is proportional to the size of D, i.e., $O(m)$. The total time-cost of recomputing the component intervals including the preprocessing is $O((m+n)\log n)$ by Theorem 3.1. The total cost of the $O(m)$ insertions from and deletions into Q, and the $O(n)$ stabbing queries for Q is $O((m+n)\frac{\log n}{\log\log n})$ by Fact 3 for $f(n) = \Theta(\frac{\log n}{\log\log n})$.

5 A Cubic-Time Algorithm for the Optimization Problem

Unfortunately, it is often impossible to construct an exact consensus tree. Therefore, the optimization version of the problem of inferring a local consensus tree in which the objective is to find a consensus tree for an as large as possible subset of an input set of 3-leaf constraints was considered in [7]. Only approximation algorithms for this optimization problem were provided in [7], and the complexity status of this problem is still open. Here we show that the problem of finding an *ordered* consensus tree for a largest possible subset of the input set of constraints of the aforementioned form is solvable by dynamic programming in cubic time.

Theorem 4. *For a set of m 3-leaf constraints on leaf-labels in $\{1, 2, ..., n\}$, we can find an ordered consensus tree for the maximum number of the constraints in time $O(n^3)$.*

Proof. For each pair of i, j, in $\{1, 2, ..., n\}$, we recursively construct the ordered consensus tree $T_{i,j}$ for the maximum number $m_{i,j}$ of the input constraints on leaf-labels in $\{i,\ i{+}1,\ .....,j\}$. For $i = 1, 2, ..., n$, the tree $T_{i,i}$ simply consists of the leaf labeled with i and its parent which is the root of the tree. To determine a tree $T_{i,j}$ where $i < j$, we compute, for $k = i, ..., j-1$, the number $w_{i,k,j}$ of the input constraints $(\{l_1, l_2\}, l_3)$ where $\{l_1, l_2, l_3\} \subset \{i, ..., j\}$ and either $\{l_1, l_2\} \subset \{i, ..., k\}$ & $l_3 \in \{k+1, ..., j\}$ or $l_3 \in \{i, ..., k\}$ & $\{l_1, l_2\} \subset \{k+1, ..., j\}$. Next, we find a $k \in \{i, ..., j\}$ which maximizes the sum $m_{i,k} + m_{k+1,j} + w_{i,k,j}$ and form the tree $T_{i,j}$ by creating its root and making the roots of $T_{i,k}$ and $T_{k,j}$ its children. It follows by induction on $|j-i|$ that $m_{i,j} = m_{i,k} + m_{k+1,j} + w_{i,k,j}$, which proves the correctness of the construction of $T_{i,j}$. Assuming that the numbers $w_{i,k,j}$ are precomputed and the trees $T_{i',j'}$ where $j' - i' < j - i$ are already determined, the determination of $T_{i,j}$ takes $O(j-i+1)$ time. The precomputation of all the numbers $w_{i,k,j}$, $i \le k \le j$, can be done by a single scan of the input constraint sequence in time $O(m + n^3)$, i.e., $O(n^3)$. The cubic upper time-bound follows.

6 Final Remarks

It would be interesting to consider the more general situation when only a partial order of the leaf labels is given *a priori*. One could look here for efficient algorithms for constructing consensus trees that would take advantage of the input partial leaf-label order as much as possible.

Our cubic-time optimization algorithm can be generalized to include other more general forms of local tree constraints, e.g., constraints of the form "the lowest common ancestor of i and j has to be a proper descendant of the lowest common ancestor of k and l" considered by Aho *et al.* in [1].

References

1. A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computing*, Vol. 10, No. 3, 1981, pp. 405–421.
2. H.J. Bandelt and A. Dress. Reconstructing the Shape of a Tree from Observed Dissimilarity Data. *Adv. Appl. Math.*, Vol. 7, 1986, pp. 309–343.
3. P. Buneman. The Recovery of Trees from Measures of Dissimilarity. In F.R. Hudson, D.G. Kendall, and P. Tautu (eds.), *Mathematics in the Archaeological and Historical Sciences*, Edinburgh Univ. Press, Edinburgh, 1971, pp. 387–395.
4. L.L. Cavalli-Sforza and A.W.F. Edwards. Phylogenetic Analysis: Models and Estimation Procedures. *Am. J. Hum. Genet.*, Vol. 19, 1967, pp. 233–257.
5. S.W. Cheng and R. Janardan. Efficient Maintenance of the Union of Intervals on a Line, with Applications. *Journal of Algorithms*, Vol. 12, 1991, pp. 57–74.
6. W.M. Fitch and E. Margoliash. Construction of Phylogenetic Trees. *Science*, Vol. 155, 1967, pp. 279–284.
7. L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Computing Evolutionary Trees. In Proc. *3rd Annual International Computing and Combinatorics Conference* (COCOON'97), LNCS 1276, Springer-Verlag, pp. 134–145.
8. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

9. D. Gusfield. Efficient Algorithms for Inferring Evolutionary Trees. *Networks*, Vol. 21, 1991, pp. 19–28.
10. M.R. Henzinger, V. King, and T. Warnow. Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Biology. In Proc. *7th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'96), pp. 333–340.
11. S. Kannan, T. Warnow, and S. Yooseph. Computing the Local Consensus of Trees. In Proc. *6th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'95), pp. 68–77.
12. N. Korte and R.H. Möhring. An incremental linear time algorithm for recognizing interval graphs. *SIAM Journal of Computing*, Vol. 18, 1989, pp. 68–81.
13. K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. EATCS Monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1984.
14. M. Overmars. Computational geometry on a grid (an overview). *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Vol. F40, 1987, pp. 167–184.
15. C. Phillips and T. Warnow. The Asymmetric Median Tree - A New Model for Building Consensus Trees, In Proc. *7th Annual Symposium on Combinatorial Pattern Matching* (CPM'96), LNCS 1075, Springer-Verlag, pp. 234–252.
16. M. Steel. The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees. *Journal of Classification*, Vol. 9, 1992, pp. 91–116.
17. M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive Evolutionary Trees. *J. Theor. Biol.*, Vol. 64, 1977, pp. 199–213.