Alcom-FT Technical Report Series
ALCOMFT-TR-01-131

Computing the Quartet Distance Between Evolutionary Trees in Time $O(n \log^2 n)$

Gerth Stølting Brodal* Rolf Fagerberg*
Christian N. S. Pedersen*

Abstract

Evolutionary trees describing the relationship for a set of species are central in evolutionary biology. Comparing evolutionary trees to quantify differences arising when estimating trees using different methods or data is a fundamental problem. In this paper we present an algorithm for computing the quartet distance between two unrooted evolutionary trees of n species in time $O(n \log^2 n)$. The previous best algorithm runs in time $O(n^2)$. The quartet distance between two unrooted evolutionary trees is the number of quartet topology differences between the two trees, where a quartet topology is the topological subtree induced by four species.

1 Introduction

The evolutionary relationship for a set of species is commonly described by an evolutionary tree. This is a rooted tree where the leaves correspond to the species, and the internal nodes correspond to specialization events, i.e. the points in time where the evolution has diverged in different directions. The direction of the evolution is described by the location of the root, which corresponds to the most recent common ancestor for all the species, and the rate of evolution is described by assigning lengths to the edges. The true evolutionary tree for a set of species is rarely known, hence estimating it from obtainable information about the species, e.g. genomic data, is of great interest. The problem of estimating aspects of the true evolutionary tree computationally requires a model describing how to use the available information about the species to solve the problem. Given a model, the problem of estimating certain aspects of the true evolutionary tree is often referred to as constructing the evolutionary tree in that model. Many models and methods for constructing evolutionary trees have been presented, see [10, Chapter 17] for an overview.

*BRICS (Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation), Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Århus C, Denmark. E-mail: {gerth,rolf,cstorm}@brics.dk. Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

An important aspect of the true evolutionary tree is the undirected tree topology induced when ignoring the location of root and the length of the edges. Many models and methods are concerned with estimating this important aspect of the true evolutionary tree, usually under the further assumption that all internal nodes have degree three. We say that such models and methods are concerned with constructing the unrooted evolutionary tree of degree three for a set of species. For the remainder of this paper an evolutionary tree denotes an unrooted evolutionary tree of degree three.

Different models and methods often yield different estimates of the evolutionary tree for the same set of species. The same model and method can also give rise to different evolutionary trees for the same set of species when applied to different information about the species, e.g. different genes. To study such differences in a systematic manner, one must be able to quantify differences between evolutionary trees using well-defined and efficient methods.

One approach for comparing two evolutionary trees is to determine a consensus tree (or forest) that reflects common traits of the two trees, e.g. the maximum agreement subtree. Much work has been concerned with developing efficient methods for computing the maximum agreement subtree of two or more evolutionary trees, see e.g. [2]. Another approach for comparing two evolutionary trees is to define a distance measure between two trees and compare the two trees by computing the distance. Several distance measures have been proposed, e.g. the symmetric difference metric [12], the nearest-neighbor interchange metric [16], the subtree transfer distance [1], the Robinson and Foulds metric [13], and the quartet metric [8]. Each distance measure has different properties and reflects different aspects of biology, e.g. the subtree transfer distance is related to the number of recombinations between the two sets of species. The quartet metric has several attractive properties. Bryant *et al.* in [5] discuss the properties of the quartet metric and conclude that it does not suffer from drawbacks of the other distance measures. For example, measures based on transformation operations, e.g. the subtree transfer distance, do not distinguish between transformations that affect a large number of leaves and transformations that affect a small number of leaves.

In this paper we study the quartet metric. For an evolutionary tree, the *quartet topology* of four species is the topological subtree induced by these species. In general, the possible quartet topologies for four species are the four shown in Fig. 1. Of these, the right-most cannot occur if we assume that all internal nodes have degree three. It is well known that the complete set of quartets is unique for a given tree and that the tree can be uniquely recovered from its set of quartets in polynomial time [6]. However, if the tree has degree three, then, as observed in [11], it can be recovered from its set of quartets in time $O(n \log n)$ using methods for constructing an evolutionary tree in the experiment model in time $O(n \log n)$ as described in [4, 9, 11].

Given two evolutionary trees on the same set of n species, the *quartet distance* between them is the number of quartet topology differences. Since there are $\binom{n}{4}$ different quartets, the quartet distance can be calculated in time $O(n^4)$ by comparing the possible quartets one by one. Steel and Penny in [14] present an algorithm for computing the quartet distance in time $O(n^3)$. Bryant *et al.* in [5] present an algorithm that computes the quartet distance in time $O(n^2)$. In this paper we present an algorithm that computes the quartet distance in time $O(n \log^2 n)$ making it possible to compare much larger evolutionary trees. Our solution is based on two techniques: the smaller-half trick, also used by methods for finding tandem repeats in strings, e.g. [15], and a data structure related to the data structure for dynamic expression trees cf. [7].

The rest of the paper is organized as follows. In Sect. 2 we introduce quartets and present our algorithm for computing the quartet distance between two unrooted evolutionary trees. In Sect. 3 we describe a hierarchical decomposition of unrooted trees which is an essential part of the data structure used by our algorithm. In Sect. 4 we present the details of our data structure.

2 The Algorithm

As mentioned, we in this paper by *evolutionary tree* mean an unrooted tree where all nodes are either leaves (i.e. have degree one) or have degree three, and where the leaves are uniquely labelled by the elements of a set S of species. Let n denote the size of S.

For an evolutionary tree T, the *quartet topology* of four species a, b, c, d is the topological subtree of T induced by these species. In general, the possible quartet topologies for species a, b, c, d are the four shown in Fig. 1. Of these, the right-most does not occur in our setting, due to the assumption about all internal nodes having degree tree. Hence, the quartet topology is a pairing of the four species into two pairs, defined by letting a and b be a pair if among the three paths in T from a to b, c, and d, the path to b is the first to separate from the others.

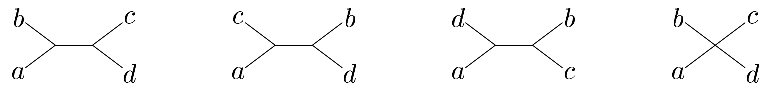


Figure 1: The four possible quartet topologies of species a, b, c, and d.

Given two evolutionary trees T_1 and T_2 on the same set S of species, the *quartet distance* between the two trees is the number of four-sets $\{a, b, c, d\} \subseteq S$, for which the quartet topologies in T_1 and T_2 differ. As there are $\binom{n}{4}$ different four-sets in S, the quartet distance can also be calculated as $\binom{n}{4}$ minus the number of four-sets for which the quartet topologies in T_1 and T_2 are identical. In this paper, we give an algorithm for finding this number in $O(n \log^2 n)$ time.

To facilitate the counting of identical quartet topologies in the two trees, we view the quartet topology of a four-set $\{a, b, c, d\}$ as two *oriented* quartet topologies given by the two possible orientations of the "middle edge" of the topology. Figure 2 shows the two oriented quartet topologies arising from one unoriented quartet topology.

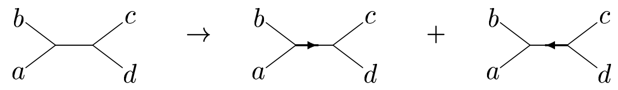


Figure 2: The two orientations of a quartet topology.

Clearly, the number of identical oriented quartet topologies between the trees T_1 and T_2 is twice the number of identical unoriented quartet topologies. The goal of our algorithm is to count identical oriented quartet topologies. For brevity, we in the rest of this paper let the word *quartet* denote an oriented quartet topology of a four-set.

We associate quartets to internal nodes in T_1 as follows: Consider the generic quartet in Fig. 3, where the orientation is from the pair $\{a, b\}$ to the pair $\{c, d\}$. There is a unique node v in T_1 where the paths from a and b to c (and d) meet. We associate the quartet of Fig. 3 with the node v. This partitions the $2\binom{n}{4}$ quartets into $n-2$ disjoint sets (as there are $n-2$ internal nodes in a tree of n leaves, when all internal nodes have degree three).

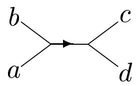


Figure 3: A generic quartet.

For an internal node v in T_1, let the three subtrees which arise if v and its three incident edges are removed be denoted by A, B, and C. The number of quartets associated with v is given by the expression

$$\binom{|A|}{2} \cdot |B| \cdot |C| + \binom{|B|}{2} \cdot |C| \cdot |A| + \binom{|C|}{2} \cdot |A| \cdot |B| \,,$$

where $|T|$ denotes the number of leaves in subtree.

The strategy of the algorithm is for each internal node v in T_1 to count how many of the quartets associated with v do also exist in T_2. The sum over all nodes in T_1 of these counts then gives the required number of identical quartets in T_1 and T_2.

The algorithm will make essential use of the data structure described in Sect. 4. The data structure maintains a coloring of the elements of S using the three colors $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. Given a pointer to an element in the data structure, its color can be changed in $O(\log n)$ time. The central feature of the data structure is the following: Let v be an internal node in T_1 with three incident subtrees A, B, C, as described above. Assume that the elements of S which are labels of leaves in A all have color $\mathcal{A}$, and that the same statement also holds for B and color $\mathcal{B}$, and for C and color $\mathcal{C}$. Then the data structure also supports that the number of quartets associated with v which also are in T_2 can be returned in $O(1)$ time. When the elements of S are colored as just described, we say that they are colored *according* to v.

The algorithm starts by rooting T_1 at an arbitrary leaf. It then calculates the size of each node in T_1 during a postorder traversal starting at the root (where the size of a node denotes the number of leaves below it), storing this information in the nodes. It also colors all elements of S by the color $\mathcal{C}$.

The algorithm then calculates the count for the internal nodes in T_1 in a recursive fashion, starting at the single child of the root of T_1. To achieve the claimed complexity, the algorithm at a node v will recurse first on its smallest child, then on its largest child, and finally add the count for v to the sum so far.

In Fig. 4, the pseudo-code for the recursive procedure, termed $\texttt{Count}(v)$, is shown. The two routines $\texttt{Small}(v)$ and $\texttt{Large}(v)$ return the smallest, respectively the largest, of the two children of an internal node v in T_1. The routine $\texttt{NodeCount}(v)$ is a call to the data structure of Sect. 4, returning the count for the node v. The routine $\texttt{ColorLeaves}(v, \mathcal{X})$ colors with the color $\mathcal{X}$ all elements in the data structure which are labels of leaves below v in T_1. This is done by a traversal of the subtree in T_1 rooted at v. By maintaining bi-directional pointers between elements of S in the data structure, and the leaves in T_1 and T_2 which they label, this takes time $O(|v| \cdot \log n)$, where $|v|$ denotes the size of v.

Procedure $\texttt{Count}(v)$
  if v is a leaf **then**
    color v by the color $\mathcal{A}$
    return 0
  $x = \texttt{Count}(\texttt{Small}(v))$
  $\texttt{ColorLeaves}(\texttt{Small}(v), \mathcal{C})$
  $y = \texttt{Count}(\texttt{Large}(v))$
  $\texttt{ColorLeaves}(\texttt{Small}(v), \mathcal{B})$
  $z = \texttt{NodeCount}(v)$
  $\texttt{ColorLeaves}(\texttt{Small}(v), \mathcal{A})$
  return $x + y + z$

Figure 4: The algorithm.

Theorem 1 *Let T_1 and T_2 be two unrooted evolutionary trees on the same set S of species, and let all internal nodes in the trees have degree three. Then the quartet distance between T_1 and T_2 can be found in $O(n \log^2 n)$ time.*

Proof. We here assume the existence of the data structure discussed above. This existence is proven in Sect. 4. We claim that the algorithm above maintains the following invariant:

1. At the *beginning* of the execution of an instance of `Count`(v), all elements in S are colored by the color $\mathcal{C}$.

2. At the *end* of the execution of an instance of `Count`(v), all elements in S which are labels of leaves *below* v in T_1 are colored by the color $\mathcal{A}$.

The claimed invariants follow by induction on the number of calls to `Count`(v). The invariants imply that when a call to `NodeCount`(v) takes place, the coloring of the elements in S are as required for the data structure – i.e., the elements labelling the leaves of the three subtrees of T_1 induced by a removal of v are colored with a different color for each of the three subtrees. Correctness of the algorithm follows.

For complexity, note that the work incurred by an instance of `Count`(v), not counting recursive calls made during this instance, is $O(1 + x \log n)$, where x denotes the size of `Small`(v). Let this work be accounted for by charging each leaf below `Small`(v) in T_1 (or v itself, if it is a leaf) an amount $O(\log n)$ of work. For a given leaf l, this charging can only happen at nodes v on the path from l to the root where the path goes from `Small`(v) to v. As the size of v is at least twice as large as the size of `Small`(v), this can only happen $\log n$ times. Hence, each leaf is at most charged $O(\log^2 n)$ work in total, and the result follows. □

3 Hierarchical Decomposition

An essential part of the data structure in Sect. 4, is a *hierarchical decomposition* of the evolutionary tree T_2. Given an unrooted tree T where all nodes have degree at most three, we in the following describe how to obtain a hierarchical decomposition of T with logarithmic height. Our decomposition is very similar to the decompositions used for solving the parallel and dynamic expression tree evaluation problems [3, 7], but in our setting the underlying tree is considered to be unrooted.

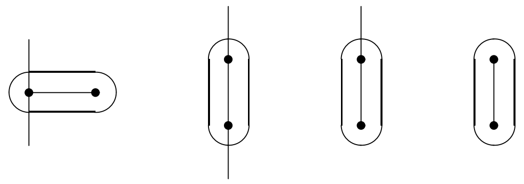
We base our hierarchical decomposition on the notion of *components*. We define a component C in T to be either

1. A single node of T, or

2. A connected subset of the nodes of T, such that at most two nodes in C are connected by an edge to nodes in $T \setminus C$.

The *external edges* of a component C of T are the edges in T connecting nodes in C and $T \setminus C$. The *degree* of a component is the number of external edges of the component. By the second condition above, a component with two or more nodes can have degree at most two.

A hierarchical decomposition of an unrooted tree T is a rooted binary tree, in the following denoted $H(T)$. There is a one-to-one mapping between the nodes of T and the leaves of $H(T)$. Each node of $H(T)$ represents a component in T. An internal node v of $H(T)$ represents the component in T that is the union of the two components represented by the two children of v. The four possible legal types of compositions of adjacent components are depicted in Fig. 5. Nodes represent contracted components and ovals possible component compositions. Types (i), (iii), and (iv) are the cases where a component with one external edge is composed with the adjacent components of degree three, two and one respectively. Type (ii) is the case where two adjacent components with degree two are composed into a new component with degree two. Note that each composition of two components corresponds to a unique edge in the tree T, namely the edge connecting the two components.



(i) (ii) (iii) (iv)

Figure 5: The four possible types of compositions of components.

Lemma 1 *For every unrooted tree with n nodes and all nodes having degree at most three, there exists a hierarchical decomposition tree with height $O(\log n)$. The decomposition can be computed in time $O(n)$.*

Proof. Given a tree with n leaves, we will construct a hierarchical decomposition bottom-up in $O(\log n)$ steps. Initially we start with each node in T being a component by itself. Let n denote the current number of components, and n_1, n_2, and n_3 the number of components of degree one, two and three respectively, i.e. $n = n_1+n_2+n_3$ for $n \geq 2$. From (i), (iii), and (iv) we have that a component with degree one can always be composed with its adjacent component. For $n \geq 5$, it holds that no degree three node is adjacent to three components with degree one. Hence, a composition of type (i), (iii), and (iv) can at most conflict with one other composition involving a component with degree one. It follows that for $n \geq 5$, at least $\lceil n_1/2 \rceil$ nonconflicting compositions can be identified if we select compositions of type (i), (iii), and (iv) greedily in time $O(n)$.

For $n \geq 2$ we have $n_1 = n_3 + 2$ and $n_2 = n - n_1 - n_3 = n - 2n_1 + 2$. For $n \geq 4$ each component with degree two is adjacent to at least one component with degree two or three. Since at most three components with degree two can be adjacent to a component with degree three, the number of components with degree two that are adjacent to a component also with degree two is at least $n_2 - 3n_3 = (n - 2n_1 + 2) - 3(n_1 - 2) = n - 5n_1 + 8$. Since each composition of type (ii) can at most conflict with two other compositions of type (ii), it follows that for $n \geq 4$, at least $\lceil (n - 5n_1 + 8)/4 \rceil$ nonconflicting compositions of type (ii) can be identified if we select the compositions greedily in time $O(n)$.

It follows that for $n \geq 5$, we can identify $\max\{\lceil n_1/2 \rceil, \lceil (n - 5n_1 + 8)/4 \rceil\} \geq n/14$ nonconflicting compositions in time $O(n)$. By repeating the above k times, at most $n(13/14)^k$ components remain. In particular, after at most $\lceil \log_{14/13}(n/4) \rceil$ steps, at most four components remain. By at most three additional compositions we have the final hierarchical decomposition. It follows that the height of the hierarchical decomposition tree is bounded by $\lceil \log_{14/13}(n/4) \rceil + 3 = O(\log n)$. Since the number of components decreases geometrically for each time we identify a set of nonconflicting compositions, the total time becomes $O(n)$. $\square$

4 Counting Quartets in Components

Given a coloring of the elements in S with the colors $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, and given a quartet oriented as in Fig. 3 from the pair $\{a, b\}$ to the pair $\{c, d\}$, we say that the quartet is *compatible* with the coloring if a and b have different colors, and c and d both have the remaining color.

Let T be an evolutionary tree for S, and let $H(T)$ be the hierarchical decomposition tree for T, as defined in Sect. 3. We now describe how to decorate the nodes of $H(T)$ with information such that the number of quartets of T which are compatible with a given coloring of S can be returned in constant time. Furthermore, for a given coloring, the information can be generated in $O(n)$ time, and if one element of S changes color, the information can be updated in $O(\log n)$ time.

For each node of $H(T)$, we store a tuple (a, b, c) of integers and a function F. Recall that a node in $H(T)$ represents a component in T. The integers a, b, and c of the tuple are the number of leaves contained in this component which are colored $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. A component has k external edges for k between zero and three (the case of zero external edges occurs only at the root of $H(T)$). The function F has three variables for each of the external edges of the component. For a component with at least one external edge, we number these edges arbitrarily from 1 to k and denote the three variables corresponding to edge i by $\boldsymbol{a}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{c}_i$. If edge i were removed from T, two subtrees of T would arise, one of them not containing the component in question. We call this the subtree *induced* by the external edge of the component. The variables $\boldsymbol{a}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{c}_i$ denote the number of leaves from the subtree induced by edge i which are colored $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. Finally, the function F states how many of the quartets associated (in the sense defined in Sect. 2) with nodes in the component are compatible with the given coloring, seen as a function of the variables $\boldsymbol{a}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{c}_i$, for $1 \leq i \leq k$. It will turn out that F is actually a polynomial of total degree at most four.

The root of $H(T)$ has no external nodes, so the function F stored there is a actually a constant. Furthermore, the root represents a component which comprises the entire tree T. Hence, the number of quartets of T which are compatible with a given coloring of S is part of the information stored at the root.

Lemma 2 *The tree $H(T)$ can be decorated with the information described above in time $O(n)$.*

Proof. The information is found in a bottom up fashion during a traversal of $H(T)$. We first describe how the information for leaves in $H(T)$, i.e. for nodes representing single node components, is generated.

For a component consisting of a single leaf colored $\mathcal{A}$, $\mathcal{B}$, or $\mathcal{C}$, the tuple clearly is $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, respectively. The function F is identically zero, as quartets are only associated with internal nodes of T, not with leaves of T.

For a component consisting of a single degree three node u, the tuple clearly is $(0,0,0)$, as no leaves of T are contained in the component. The function F should count the number of quartets which are compatible with the coloring *and* which are associated with u in T. A quartet oriented from the pair $\{a,b\}$ to the pair $\{c,d\}$ fulfills this requirement exactly if c and d are contained in the same subtree induced by an external edge of the component, and they have the same color, and a and b each are in one of the remaining two induced subtrees and each have one of the remaining two colors. Assuming that c and d are in the subtree induced by edge number one, and have color $\mathcal{A}$, the number of possible quartets fulfilling this is

$$\binom{\boldsymbol{a}_1}{2} \cdot (\boldsymbol{b}_2\boldsymbol{c}_3 + \boldsymbol{b}_3\boldsymbol{c}_2)\,.$$

Summing over all $3 \cdot 3 = 9$ choices of the induced subtree and color for c and d, we get:

$$\begin{aligned} &F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2,\boldsymbol{a}_3,\boldsymbol{b}_3,\boldsymbol{c}_3)\\ &\quad= \tbinom{\boldsymbol{a}_1}{2}\cdot(\boldsymbol{b}_2\boldsymbol{c}_3+\boldsymbol{b}_3\boldsymbol{c}_2) + \tbinom{\boldsymbol{a}_2}{2}\cdot(\boldsymbol{b}_1\boldsymbol{c}_3+\boldsymbol{b}_3\boldsymbol{c}_1) + \tbinom{\boldsymbol{a}_3}{2}\cdot(\boldsymbol{b}_2\boldsymbol{c}_1+\boldsymbol{b}_1\boldsymbol{c}_2)\\ &\quad+ \tbinom{\boldsymbol{b}_1}{2}\cdot(\boldsymbol{a}_2\boldsymbol{c}_3+\boldsymbol{a}_3\boldsymbol{c}_2) + \tbinom{\boldsymbol{b}_2}{2}\cdot(\boldsymbol{a}_1\boldsymbol{c}_3+\boldsymbol{a}_3\boldsymbol{c}_1) + \tbinom{\boldsymbol{b}_3}{2}\cdot(\boldsymbol{a}_2\boldsymbol{c}_1+\boldsymbol{a}_1\boldsymbol{c}_2)\\ &\quad+ \tbinom{\boldsymbol{c}_1}{2}\cdot(\boldsymbol{b}_2\boldsymbol{a}_3+\boldsymbol{b}_3\boldsymbol{a}_2) + \tbinom{\boldsymbol{c}_2}{2}\cdot(\boldsymbol{b}_1\boldsymbol{a}_3+\boldsymbol{b}_3\boldsymbol{a}_1) + \tbinom{\boldsymbol{c}_3}{2}\cdot(\boldsymbol{b}_2\boldsymbol{a}_1+\boldsymbol{b}_1\boldsymbol{a}_2) \end{aligned}$$

We now turn to the generation of the information stored in the internal nodes of $H(T)$. Consider the component composition of of two components C' and C''. Let (a',b',c') and F', and (a'',b'',c'') and F'' be the information stored at the nodes representing the components C' and C''. The information stored at the node representing the composition C of C' and C'' is $(a'+a'', b'+b'', c'+c'')$ and F, where F depends on the type of composition. If the component composition is of type (ii) we consider the case where the first external edge of C' and C'' in the edge connecting C' and C'', and the second external edge of C' is the first external edge of C and the second external edge of C'' is the second external edge of C. The remaining cases of numbering the external edges are obtained by appropriate permutations of the arguments to F' and F''.

$$\begin{aligned} &F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)\\ &\quad= F'(\boldsymbol{a}_2+a'',\boldsymbol{b}_2+b'',\boldsymbol{c}_2+c'',\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1)\\ &\quad+ F''(\boldsymbol{a}_1+a',\boldsymbol{b}_1+b',\boldsymbol{c}_1+c',\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2) \end{aligned}$$

Component compositions of type (iii) and (iv) are identical to type (ii), except that the definition of F is simpler. For type (iii) we have (assuming that C'' is the component of degree one)

$$F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1) \;=\; F'(a'',b'',c'') \;+\; F''(\boldsymbol{a}_1+a',\boldsymbol{b}_1+b',\boldsymbol{c}_1+c')\;,$$

and for type (iv) we have

$$F \;=\; F'(a'',b'',c'') \;+\; F''(a',b',c')\;.$$

Note that for type (iv) compositions F is a constant. Finally, we for type (i) compositions get the following contribution assuming C' has degree one and the two external edges of C are the second and third external edge of C'' respectively.

$$\begin{aligned}
&F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)\\
&\quad= \;\; F'(\boldsymbol{a}_1+\boldsymbol{a}_2+a'',\boldsymbol{b}_1+\boldsymbol{b}_2+b'',\boldsymbol{c}_1+\boldsymbol{c}_2+c'')\\
&\quad+ \;\; F''(a',b',c',\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)
\end{aligned}$$

By structural induction on the definition of the F function in the contribution of a component, it follows that F is a polynomial of total degree at most four. Polynomials with total degree at most four and at most nine variables can be stored in constant space by storing the coefficients of the polynomials, and they can be manipulated in constant time, e.g. the addition and composition of two polynomials. We conclude that the contribution of a component C, that is composed of two components C' and C'', can be computed in constant time, provided that the contribution of C' and C'' are known, i.e. $H(T)$ can be decorated in $O(n)$ time. □

Lemma 3 *The decoration of $H(T)$ can be updated in $O(\log n)$ time when changing the color of an element in S.*

Proof. From the proof of Lemma 2 we know that the decoration of a node in $H(T)$ only depends on the decoration of the children of the node in $H(T)$, i.e. the only decorations that need to be updated in $H(T)$ while changing the color of an element in S are the ancestors of the leaf in $H(T)$ corresponding to the element. Since $H(T)$ has height $O(\log n)$ and the decoration of a node takes constant time to compute knowing the decoration of the children, it follows that the decoration of $H(T)$ can be updated in time $O(\log n)$. □

Lemma 4 *When S is colored according to a choice of v in T_1, then the set of quartets compatible with the coloring is exactly the quartets associated with v.*

Proof. Follows from the definitions of the colors and compatible quartets. □

Corollary 1 *If the above construction is done with T_2 for T, and the coloring of S is according to a choice of v in T_1, then the quartets in T_2 compatible with the coloring are exactly the quartets which are in both T_1 and T_2. Furthermore, the number of such quartets is exactly the value of the constant function F stored at the root of $H(T_2)$.*

References

[1] B. L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:1–13, 2001.

[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] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, Apr. 1974.

[4] G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. In *Proc. 28th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[5] D. Bryant, J. Tsang, P. E. Kearney, and M. Li. Computing the quartet distance between evolutionary trees. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 285–286, N.Y., Jan. 9–11 2000. ACM Press.

[6] P. Buneman. The recovery of trees from measures of dissimilairty. *Mathematics in Archeological and Historial Sciences*, pages 387–395, 1971.

[7] R. F. Cohen and R. Tamassia. Dynamic expression trees. *Algorithmica*, 13(3):245–265, 1995.

[8] G. Estabrook, F. McMorris, and C. Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Syst. Zool.*, 34(2):193–200, 1985.

[9] M. Farach, S. Kannan, and T. J. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, 1995.

[10] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[11] A. Lingas, H. Olsson, and A. Östlin. Efficient merging, construction, and maintenance of evolutionary trees. In *Proc. 26th Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 544–553. Springer-Verlag, 1999.

[12] D. F. Robinson and L. R. Foulds. Comparison of weighted labelled trees. In *Combinatorial mathematics, VI (Proc. Sixth Austral. Conf., Univ. New England, Armidale, 1978)*, Lecture Notes in Mathematics, pages 119–126. Springer, Berlin, 1979.

[13] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Math. Biosci.*, 53(1-2):131–147, 1981.

[14] M. Steel and D. Penny. Distribution of tree comparison metrics–some new results syst. *Syst. Biol.*, 42(2):126–141, 1993.

[15] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. In *Proc. 9th Annual Symposium on Combinatorial Pattern Matching*, volume 1448 of *Lecture Notes in Computer Science*, pages 140–152. Springer-Verlag, 1998.

[16] M. S. Waterman and T. F. Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73:789–800, 1978.