COMPUTING THE QUARTET DISTANCE BETWEEN EVOLUTIONARY TREES OF BOUNDED DEGREE

M. STISSING, C. N. S. PEDERSEN, T. MAILUND[*] AND G. S. BRODAL

Bioinformatics Research Center, and Dept. of Computer Science, University of Aarhus, Denmark

R. FAGERBERG

Dept. of Mathematics and Computer Science, University of Southern Denmark, Denmark

We present an algorithm for calculating the quartet distance between two evolutionary trees of bounded degree on a common set of n species. The previous best algorithm has running time $O(d^2 n^2)$ when considering trees, where no node is of more than degree d. The algorithm developed herein has running time $O(d^9 n \log n))$ which makes it the first algorithm for computing the quartet distance between non-binary trees which has a sub-quadratic worst case running time.

1. Introduction

The evolutionary relationship between a set of species is conveniently described as a tree, where the leaves represent the species and the inner nodes speciation events. Using different biological data, or different methods of inferring such trees (see e.g. Felsenstein[1] for an overview) can yield different inferred trees for the same set of species, and to study such differences in a systematic manner, one must be able to quantify such differences using well-defined and efficient methods. Several distance measures have been proposed,[2–6] each having different properties and reflecting different aspects of biology.

This paper concerns efficient computation of the quartet distance,[3] a distance measure with several attractive properties.[7,8] For an evolutionary tree, the *quartet topology* of four species is determined by the minimal topological subtree containing the four species. The four possible quartet topologies of four species are shown in Fig. 1. The three leftmost of these we denote *butterfly* quartets, the rightmost is a *star* quartet. Given two evolutionary trees on the same set of n species, the *quartet distance* between them is the number of sets of four species for which the quartet topologies differ in the two trees.

For binary trees, the fastest method for computing the quartet distance between two trees runs in $O(n \log n)$[9], but for trees of arbitrary degree, the fastest algorithms run in $O(n^3)$ (independent of the maximal degree) or $O(n^2 d^2)$ (where d is the maximal degree in the tree)[10]. This paper focuses on trees where each inner node v has degree at most d, where d is a fixed constant. We develop an $O(d^9 n \log n)$ time and $O(d^8 n)$ space algorithm for

---

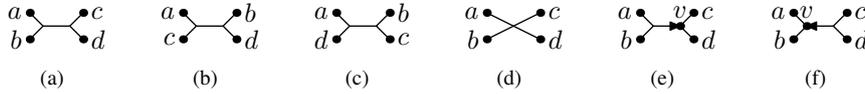[*]Current affiliation: Dept. of Statistics, University of Oxford, UK

2

Figure 1.   Figures (a)–(d) show the four possible quartet topologies of species a, b, c, and d. Figures (e) and (f) show the two ordered butterfly quartet topologies induced by the butterfly quartet topology in (a).

computing the quartet distance between such two trees, based on the algorithm in Brodal et al.[9] This is the first algorithm for computing the quartet distance between non-binary trees with a sub-quadratic worst case running time. In Brodal et al.[9] the quartet distance was calculated as $\binom{n}{4}$ minus the number of shared quartets. We will adopt this approach, focusing on calculating shared quartets, noting that in our setting trees might include star quartets. We first consider calculating the number of shared butterfly quartets between two trees, and then extend the algorithm into calculating shared star quartets as well.

2. Terminology

An *evolutionary tree* is an unrooted tree where any node, v, is either a leaf or an inner node of degree d_v, where $3 \leq d_v \leq d$. Leaves are uniquely labeled by the elements of a set S of species, where $|S| = n$. For an evolutionary tree T, the *quartet topology* of a set $\{a, b, c, d\} \subseteq S$ of four species is the topological subtree of T induced by these species. The possible quartet topologies for species a, b, c, d are shown in Fig. 1. An evolutionary tree with n leaves gives rise to $\binom{n}{4}$ different quartet topologies. Butterfly quartet topologies are a pairing of the four species into two pairs, defined, see Fig. 1, by letting a and b be a pair if the path from a to b doesn't meet the path from c to d. We view the (butterfly) quartet topology of a four-set of species $\{a, b, c, d\}$ as two *oriented quartet topologies*[9], given by the two possible orientations of the middle edge of the topology, see Fig. 1. An oriented quartet topology is thus an ordered pair of two-sets, e.g. $(\{a, b\}, \{c, d\})$. The number of oriented quartet topologies of a tree is twice the number of unoriented quartet topologies. In the rest of this paper, until Sect. 6 we by *quartet* consider an oriented quartet topology and use the notation $ab|cd$ for $(\{a, b\}, \{c, d\})$.

Let Q be the set of all possible quartets of S. Let $Q_T \subset Q$ denote the set of quartets in an evolutionary tree T. We will *associate* quartets of Q to inner nodes v of T, such that $ab|cd$ is associated to v if v is the node where the paths from c to a and d to a meet (see Fig. 1, right hand side). In the terminology of Christiansen et al.[10] these are all the quartets claimed by edges pointing to v. By Q_v we denote the set of all quartets associated to v. Having the trees incident to v, $T_1, T_2, \ldots, T_{d_v}$, see Fig. 2, a quartet $ab|cd$ is associated to v if and only if a and

Figure 2. An inner node $v \in T$ with incident subtrees $T_1, \ldots, T_6$

b are in the same subtree and c and d are in two different subtrees. The total number of quartets associated to v, $|Q_v|$ is then $\sum_i \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}} \binom{|T_i|}{2} |T_j||T_k|$ where i, j, and k is in the interval $1 \ldots d_v$, and $|T|$ denotes the number of leaves in T and denote this the *size* of
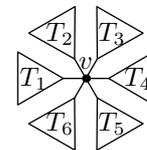
T. The main strategy of finding the shared quartets between two trees, T and T', is, for each v in T, to count how many of the quartets associated with v are also quartets of T' and calculate the sum over all v, $\sum_{v \in T} |Q_v \cap Q_{T'}|$. Doing this we will relate quartets to a colouring, using the d colours $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{d-2}, \mathcal{C}$, of the elements in S. For an internal node v in T, we will say that S is coloured *according* to v if all leaves in each subtree incident to v is coloured using one colour and no other subtree has its leaves coloured this colour. Having a colouring of S and a quartet $ab|cd$, we say that the quartet is *compatible* with the colouring if c and d have different colours and a and b have a third colour. These, almost identical, definitions gives us the following lemma, similar to Brodal et al.[9], Lemma 1.

Lemma 2.1. *When S is coloured according to a choice of v in T, the set of possible quartets compatible with the colouring is exactly the set Q_v of quartets associated with v.*

Consequently, if S is coloured according to v in T, the quartets in $Q_{T'}$ compatible with this colouring are exactly the quartets associated with v that are also quartets of T'. The algorithm will, for each v in T, ensure a colouring according to v and then count the number of quartets of T' compatible with this colouring. In order to do this colouring, we will maintain pointers between elements of S and the leaves of T and T' and vice versa.

3. The Basic Algorithm — $O(d^9 n \log^2 n)$

In this section we expand the idea given above into an algorithm for calculating the shared quartets between T and T' with running time $O(d^9 n \log^2 n)$. The algorithm colours S according to nodes v (using the procedure `colourLeaves(U, X)`, which colours all leaves in U with the colour $\mathcal{X}$) and uses a hierarchical decomposition tree $H_{T'}$ in counting the number of quartets in T' compatible with this colouring, `shared(v, T')`. The hierarchical decomposition tree, described in detail in Sect. 5, enables a change of colour of k leaves in time $O(d^9(k + k \log \frac{n}{k}))$ and achieves $O(1)$ time for calculating `shared(v, T')`. The hierarchical decomposition tree $H_{T'}$ is constructable in $O(d^8 n)$ time and $O(d^8 n)$ space.

A pseudocode version of the algorithm is given in Alg. 1. The algorithm assumes T has been rooted in an arbitrary leaf. Let $|v|$ denote the number of leaves in the subtree rooted at v, and call this the *size* of v. A simple traversal lets us annotate each node v such that it knows its largest child, $Large(v)$—where in case of a tie we arbitrarily select one—and which of its children are not the largest, $Small_i(v)$. Let $Small_i(v)$ denote the i'th smallest subtree, with respect to the number of leaves in this subtree. Prior to the first call of the algorithm, the root of T is coloured $\mathcal{C}$ and all (other) leaves are coloured $\mathcal{A}$. The algorithm is initially called with the single child of the root of T. The algorithm recurses through the entire tree, summing the number of shared quartets between v and T', $\sum_{v' \in T'} |Q_v \cap Q_{v'}|$, for each v, ultimately calculating $|Q_T \cap Q_{T'}|$. The algorithm colours the leaves according to v and then counts the number of shared quartets. It then recurses, first on the largest child of v, $Large(v)$ and then on the smaller children of v, $Small_i(v)$. Before recursing on a node v the algorithm ensures that all leaves below v are coloured $\mathcal{A}$. Returning from the recursion, the algorithm ensures that any leaf below v is coloured $\mathcal{C}$.

4

Algorithm 1 $\mathtt{count}(v, T')$ - *count number of shared butterfly quartets between the subtree rooted at v and T'*

Require: v a non root node of T, all leaves below v is coloured $\mathcal{A}$, all leaves not in v coloured $\mathcal{C}$.
Ensure: Res is the no. of quartets shared between nodes in v and T'. All leaves in v are coloured $\mathcal{C}$.

  if v is a leaf **then**
        $\mathtt{colourLeaves}(v, \mathcal{C})$
        $Res \leftarrow 0$
  else
        $Res \leftarrow 0$
        for all $Small_i(v)$ **do**
            $\mathtt{colourLeaves}(Small_i(v), \mathcal{B}_i)$
        $Res \leftarrow Res + \mathtt{shared}(v, T')$
        for all $Small_i(v)$ **do**
            $\mathtt{colourLeaves}(Small_i(v), \mathcal{C})$
        $Res \leftarrow Res + \mathtt{count}(Large(v))$
        for all $Small_i(v)$ **do**
            $\mathtt{colourLeaves}(Small_i(v), \mathcal{A})$
            $Res \leftarrow Res + \mathtt{count}(Small_i(v))$
  return Res

We see that the algorithm colours a leaf only when this leaf is in a smaller subtree, $Small_i(v)$, of some v on which $\mathtt{count}(v)$ is invoked. As v is at least twice the size of any $Small_i(v)$, any leaf can at most be coloured $O(\log n)$ times. As the hierarchical decomposition tree enables the change of colour of k leaves in time $O(d^9(k + k \log \frac{n}{k})) \subseteq O(d^9 k \log n)$, we can charge this by letting each colouring of a leaf be of $O(d^9 \log n)$ cost. The entire algorithm, as the colouring is the predominant time consuming factor, is then of time $O(d^9 n \log^2 n)$. The space used is dominated by the space used by the hierarchical decomposition tree, which is $O(d^8 n)$ cf. Sect. 5.

4. The Improved Algorithm — $O(d^9 n \log n)$

The analysis of the basic algorithm above shows that if any node v "uses" time $O(d^9 \log n \cdot \sum_i |Small_i(v)|)$ then the entire algorithm uses time $O(d^9 n \log^2 n)$. This is often referred to as the smaller-half trick:

Lemma 4.1. (Smaller-half trick) *If any inner node v supplies a term $c_v = c \cdot \sum_i |Small_i(v)|$ and any leaf a term $c_v = 0$, then the sum over all nodes $\sum_v c_v \leq c \cdot n \log n$.*

This is easily proved by induction. As an instance of this, the analysis above used $c = d^9 \log n$. The improved algorithm below, uses an extended smaller-half trick which is also easily proved by induction.

Lemma 4.2. (Extended smaller-half trick) *In a rooted tree, if any inner node v supplies a term $c_v = c \cdot \sum_i |Small_i(v)| \log \left(\frac{|v|}{|Small_i(v)|} \right)$ and any leaf a term $c_v = 0$, then the sum over all nodes $\sum_v c_v \leq c \cdot n \log n$.*

The main observation in achieving the improved algorithm comes from noting that, whenever the basic algorithm $\mathtt{count}(v)$ is called, all leaves outside the subtree rooted at v will have the colour $\mathcal{C}$ and these leaves will not change their colour while $\mathtt{count}(v)$ is being processed. This, of course, also applies to the leaves of T' coloured $\mathcal{C}$. We will therefore, in certain cases, construct a compact representation of T', by "contracting" nodes

---

Algorithm 2 `fastCount`$(v,\ T')$ – *count number of shared butterfly quartets between the subtree rooted at v and T'*

Require: v a non root node of T, all leaves in v coloured $\mathcal{A}$, all leaves not in v coloured $\mathcal{C}$.
Ensure: Res equals the number of quartets shared between v and T'. All leaves in v are coloured $\mathcal{C}$.

  $Res \leftarrow 0$
  if v is a leaf **then**
      `colourLeaves`$(v, \mathcal{C},\ T')$
  else
      for all $Small_i(v)$ **do**
          `colourLeaves`$(Small_i(v), \mathcal{B}_i,\ T')$
      $Res \leftarrow Res +\ $`shared`$(v,\ T')$
      for all $Small_i(v)$ **do**
          `colourLeaves`$(Small_i(v), \mathcal{C},\ T')$
      for all $Small_i(v)$ **do**
          $T'_i \leftarrow\ $`contract`$(Small_i(v),\ $`extract`$(Small_i(v),\ T'))$
      if $|T'| > 5|Large(v)|$ **then**
          $T' \leftarrow\ $`contract`$(Large(v),\ T')$
      $Res \leftarrow Res +\ $`fastCount`$(Large(v),\ T')$
      for all $Small_i(v)$ **do**
          `colourLeaves`$(Small_i(v), \mathcal{A},\ T'_i)$
          $Res \leftarrow Res +\ $`fastCount`$(Small_i(v),\ T'_i)$
  return Res

---

of T' coloured $\mathcal{C}$. We will consider any constructed T' as having an associated hierarchical decomposition tree $H_{T'}$, see below. A pseudocode version of the improved algorithm is given in Alg. 2. If we ensure that T' (and thus $H_{T'}$) is of size $O(|v|)$ whenever `fastCount`(v, T') is processed, we know that k leaves can have their colour updated in time $O(d^9(k + k \log \frac{|v|}{k}))$. The extended smaller-half trick then ensures that the total time spent colouring is $O(d^9 n \log n)$.

The algorithm resembles the basic algorithm except for `contract`(U, Y) and `extract`(U, Y), the details of which are given in Sect. 5. For the analysis of the improved algorithm it suffices to note that `contract`(U, Y) makes a compact representation of Y by contracting anything in Y except the leaves in U. This yields a tree with no more than $4|U|$ nodes in time $O(d^9|Y|)$. Likewise `extract`(U, Y) makes a compact representation of Y. This representation also lets the leaves of U in Y remain intact. All other nodes are contracted. The leaves of the arising tree are (implicitly) coloured $\mathcal{C}$. The operation `extract`(U, Y) completes in $O(d^9|U| \log \frac{|Y|}{|U|})$ time and yields a tree of size $O(d|U| \log \frac{|Y|}{|U|})$. When constructing such a new tree, as a result of `contract`(U, Y), we will update the pointers of S to point to the leaves of the newly created tree. This manipulation of S enables the colouring of leaves in the newly created trees.

Regarding correctness, assuming the leaves in v are coloured $\mathcal{A}$ and the leaves outside v are coloured $\mathcal{C}$ when `fastCount`(v, T') is called, the algorithm will, as the basic algorithm, ensure a colouring according to v prior to the call `shared`(v, T'). Furthermore, before recursing on $Small_i(v)$ (or $Large(v)$) the algorithm ensures that the tree used in the recursion is coloured such that all leaves in $Small_i(v)$ ($Large(v)$) are coloured $\mathcal{A}$ and the leaves outside are coloured $\mathcal{C}$. The correctness of the algorithm follows from the correctness of the basic algorithm.

For time complexity, we see that $|T'|$ is of size $O(|v|)$ when `fastCount`(v, T') is called. This implies that the trees T'_i are each of size $O(|Small_i(v)|)$. The time used in con-

6

structing these is $O(d^9 \sum_i |Small_i(v)| \log \frac{|v|}{|Small_i(v)|})$, i.e. construction time is dominated by the time taken colouring the leaves colourLeaves$(Small_i(v), \mathcal{B}_i, T')$. We note that each $H_{T'_i}$ is constructable in time $O(d^8|T'_i|)$, see Sect. 5, i.e. is dominated by the time used obtaining T'_i by contraction. Contracting the larger part of T', contract$(Large(v), T')$, completes in time $O(d^9|T'|)$ and yields a tree of size at most $4|Large(v)|$ (see Lemma 5.5 below). The total time spent on repeatedly contracting larger parts of T', as we only do this when $5 \cdot |Large(v)| \le |T'|$ is thus bounded by the sum of the geometric series $\frac{4}{5}^k$ times d^9. This implies that the time spent contracting T' is linear in the initial size of T' (times d^9), i.e. the time spent is bounded by the time constructing T', the time used by contract$($extract$(Small_i(v), T'))$. Ultimately this implies that the algorithm completes in $O(d^9 n \log n)$ time.

Regarding the space used by the algorithm, we see that the only additional space it consumes is when creating T'_i's (and corresponding $H_{T'_i}$'s) at each node $v \in T$; in total no more than the maximal space used on any root-to-leaf path P_j in T, i.e. $O(d^8 \max_j \sum_{v \in P_j} \sum_i |Small_i(v)|)$. Consider a path P_j, there will be a number of nodes v, where both v and $Small_i(v)$ are on the path. The total space consumed by all such v is no more than $d^8 \frac{d-1}{d} n \sum_i \frac{1}{2^i} \in O(d^8 n)$, that is we store at most $\frac{d-1}{d}$ parts of what is left, i.e. all the smaller children, and as we know $Small_i(v)$ is on the path, we can cut of at least half. The "rest" of the path consists of pairs v and $Large(v)$. For each such pair we consume $d^8 \sum_i |Small_i(v)|$ space, we might think of this as marking the leaves in each of the $Small_i(v)$. However as no other pair v, $Large(v)$ can mark an already marked leaf, we conclude that these pairs consume $O(d^8 n)$ space. In total $O(d^8 n)$ space is used.

5. Hierarchical Decomposition Tree

The algorithms developed uses the hierarchical decomposition tree heavily. The data structure can, in constant time, calculate the number of quartets in an evolutionary tree T compatible with the current colouring of S. The data structure allows a change of the colour of k elements of S in time $O(d^9(k + k \log \frac{n}{k}))$ where n is the number of leaves in T. In the following we describe how to build and update
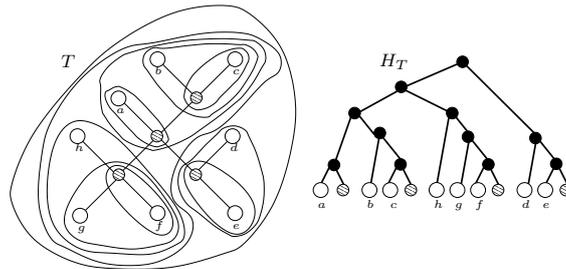

Figure 3.   A tree T and a hierarchical decomposition of this tree. H_T is the hierarchical decomposition tree corresponding to the shown hierarchical decomposition of T.

such a tree inspired by the approach in Brodal et al. [9]

The *hierarchical decomposition* of T is based on the notion of components. A component C of T is a connected subset of nodes in T. An *external edge* of C is connecting a node in C with a node outside of C. The number of external edges is the *degree* of C. We will allow two types of components: (1) *Simple components* containing a single node of T, see Fig. 4(a), 4(b). (2) *Composite components* composing two other components, where both
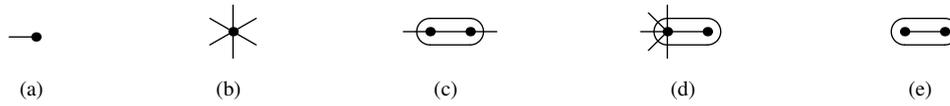
Figure 4.   Possible components: (a), (b) Simple components, a leaf and an inner node component respectively. (c) – (e) Composite components: (c) Composing two components of degree two. (d) Composing a component of degree d_C with a component of degree one. (e) Composing two degree one components – as seen, a special case of (d).

of these are of degree two 4(c) or at least one of these are of degree one, see Fig. 4(d), 4(e). Letting each node of T be a component by itself, a hierarchical decomposition of T is a set of components created by repeatedly composing these. Note that the degree of a composite component will be no more than the maximum degree of the components it is composed of. In decomposing T, note that, the current set of components form a tree, hence there will always be at least one component of degree 1, and we can therefore always continue composing until we are left with a component containing all simple components of T.

Having a hierarchical decomposition of T including a component containing all simple components of T, we might in a natural way view this as a tree. A hierarchical decomposition tree H_T for T is a rooted binary tree with leaves corresponding to simple components of T and inner nodes corresponding to composite components (components in a hierarchical decomposition of T), see Fig. 3. An inner node v, with children v' and v'', corresponds to the component C arising when the two components C' and C'', corresponding to the children of the node, are composed. The root, r, corresponds to a component containing all simple components of T. In this sense many hierarchical decomposition trees exist. We will show how to construct a *locally-balanced* hierarchical decomposition tree. A rooted binary tree with n nodes is *c-locally-balanced* if for all nodes v in the tree, the height of the subtree rooted at v is at most $c \cdot (1 + \log |v|)$, where $|v|$ is the number of leaves in the subtree and the height is the maximal number of edges on any root-to-leaf path. The following lemma is an extension of Brodal et al.[9], Lemma 3.

Lemma 5.1. *For any unrooted tree T with n nodes of degree at most d, a $6d$-locally balanced hierarchical decomposition tree H_T can be computed in time $O(dn)$.*

The following lemma from Brodal et al.[9] bounds the number of nodes on k root-to-leaf paths in a hierarchical decomposition tree.

Lemma 5.2. *The union of k root-to-leaf paths in a c-locally balanced rooted binary tree with n leaves contains at most $k(3 + 4c) + 2ck \log \frac{n}{k}$ nodes.*

Having an evolutionary tree T with n leaves and the associated hierarchical decomposition tree H_T we want to count the number of quartets in T compatible with the current colouring of S in constant time. Further, when k elements of S change their colour, we should handle this update in time $O(d^9(k + k \log \frac{n}{k}))$.

We will associate functions and vectors to the nodes of H_T. At any node v, having the associated component C, in H_T the vector $\mathbf{c} = (c_1, c_2, \ldots, c_d)$ stored holds the number of

8

leaves contained in C of colours $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{d-2}$ and $\mathcal{C}$ respectively. If C is of degree d_C, the function F stored at v, is a function of d_C vector variables. The function counts the number of quartets associated to any node in C compatible with the current colouring of S. This implies that the function stored at the root of H_T counts the total number of quartets in T compatible with the current colouring of S. Furthermore, since the component associated to the root of H_T has 0 external edges, the function stored here is a constant. The elements $\mathbf{c}^{\mathbf{i}}_{i'}$ of the vector variables $\mathbf{c}^{\mathbf{i}}$ of F correspond to the number of leaves coloured with the i''th colour in the component incident to the i'th external edge of C.

First we describe how to associate the vectors and functions to the leaves of H_T, that is the simple components of T. If v has an associated component of degree 1, i.e. it represents a leaf l in T, having the colour $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{d-2}$ or $\mathcal{C}$ the vector stored at v is $(1, 0, \ldots, 0, 0), (0, 1, \ldots, 0, 0), \ldots, (0, 0, \ldots, 1, 0)$ or $(0, 0, \ldots, 0, 1)$ respectively. Since the number of quartets associated to l is 0, the function stored at v is identically zero: $F(\mathbf{c}^{\mathbf{1}}) = 0$. Otherwise, if v, with associated component C of degree d_C, represents an internal node u in T the tuple stored here is $(0, 0, \ldots, 0, 0)$ as the component contains no leaves of any colour. The function F stored here, counts the number of quartets associated to u compatible with the colouring of S. Recall that a quartet $ab|cd$ associated to u has a and b in the same subtree incident to u and c and d in two different subtrees. Further, if $ab|cd$ is compatible with the colouring of S, a and b have the same colour and c and d have two different colours. F is then:

$$F(\mathbf{c}^{\mathbf{1}}, \mathbf{c}^{\mathbf{2}}, \ldots, \mathbf{c}^{\mathbf{d_C}}) = \sum_{i} \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}}^{d_C} \sum_{i'}^{d} \sum_{j' \neq i'}^{d} \sum_{\substack{k' \neq i' \\ k' \neq j'}}^{d} \binom{\mathbf{c}^{\mathbf{i}}_{i'}}{2} \mathbf{c}^{\mathbf{j}}_{j'} \mathbf{c}^{\mathbf{k}}_{k'}$$

We now turn to the tuples and functions associated to the inner nodes of H_T. The inner node v, with children v' and v'', will store the vector $\mathbf{c}' + \mathbf{c}''$, assuming v' and v'' store the vector $\mathbf{c}'$ and $\mathbf{c}''$ respectively. Letting F' and F'' be the functions stored at v' and v'', we express F stored at v. Let C be the component corresponding to v, likewise for C' and C''. If both C' and C'' are degree 2 components (Fig. 4(c)) we construct F as $F(\mathbf{c}^{\mathbf{1}}, \mathbf{c}^{\mathbf{2}}) = F'(\mathbf{c}^{\mathbf{1}}, \mathbf{c}^{\mathbf{2}} + \mathbf{c}'') + F''(\mathbf{c}^{\mathbf{1}} + \mathbf{c}', \mathbf{c}^{\mathbf{2}})$, assuming the second external edge of C' is the first external edge of C'' and the first 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 (other edge "numberings" are handled similarly). If C' is a component of degree $d_{C'} \geq 2$ and C'' a component of degree 1 (Fig. 4(d)), we construct F, this time assuming the $d_{C'}$'th external edge of C' is the first (and only) external edge of C'', the d_C external edges of C correspond to the d_C first external edges of C': $F(\mathbf{c}^{\mathbf{1}}, \mathbf{c}^{\mathbf{2}}, \ldots, \mathbf{c}^{\mathbf{d_C}}) = F'(\mathbf{c}^{\mathbf{1}}, \mathbf{c}^{\mathbf{2}}, \ldots, \mathbf{c}^{\mathbf{d_C}}, \mathbf{c}'') + F''(\mathbf{c}^{\mathbf{1}} + \mathbf{c}^{\mathbf{2}} + \ldots + \mathbf{c}^{\mathbf{d_C}} + \mathbf{c}')$. As a special case of the above, if both C' and C'' are of degree 1 (Fig. 4(d)), we note that F is a constant: $F = F'(\mathbf{c}'') + F''(\mathbf{c}')$. If C is a simple component, F is a polynomial of degree at most 4 with no more than $d \cdot d_C \leq d^2$ variables. By induction in the way F's are constructed, this is then seen to hold for any component. At any node v we observe that the F (and $\mathbf{c}$) to be stored is constructable in $O(d^8)$ time. This implies the following lemma, similar to Brodal et al.[9], Lemma 5:

Lemma 5.3. *The tree H_T can be decorated with the information described above in time and space $O(d^8 n)$.*

The following lemma, similar to Brodal et al.[9], Lemma 6, arises as a consequence of Lem. 5.2 and the fact that the decoration stored at a node v in H_T is constructable in $O(d^8)$ time, given the decoration at its children.

Lemma 5.4. *The decoration of H_T can be updated in $O(d^9(k + k \log \frac{n}{k}))$ time when the colour of k elements in S changes.*

The above results imply the running time of the basic algorithm. We now turn to the details of `contract` and `extract` used in the improved algorithm. The procedure `contract`(U, Y) yields a tree Y' of size $O(|U|)$ in time $O(d^9|Y|)$ letting the leaves present in U remain untouched in Y'. This is accomplished by copying Y and contracting edges corresponding to legal compositions. This way Y' contains nodes corresponding to simple or composite components. The functions and vectors stored at these components is inherited by the nodes they correspond to. Y''s edges are a subset of the edges of Y, namely the edges not contracted. The following lemma, an extension of Brodal et al.[9], Lemma 4), ensures that Y' has no more than $4|U|$ nodes, and that each of the leaves in U is a leaf in Y'.

Lemma 5.5. *Let T be an unrooted tree with n nodes of degree at most d, and let $k \geq 0$ leaves be marked as* non-contractible. *In $O(dn)$ time a decomposition of T into at most $4k$ components can be computed such that each marked leaf is a component by itself.*

Creating the information to be stored at the nodes of Y' uses $O(d^8)$ time per contraction made, that is `contract`(U, Y) completes in time $O(d^9|Y|)$. We can calculate $H_{Y'}$, in the time stated, as each node of Y' has an associated vector and function. Likewise `extract`(U, Y) yields a contracted tree Y' of size $O(d|U| \log \frac{|Y|}{|U|})$ in time $O(d^9|U| \log \frac{|Y|}{|U|})$. This is achieved by using the hierarchical decomposition tree H_Y. We mark the internal nodes of H_Y on the $|U|$ root-to-leaf paths leading to the leaves in U. Doing this bottom-up, one leaf at a time, we can stop marking when an already marked node is encountered. Lem. 5.2 then bounds the number of marked nodes. Removing all these marked nodes yields a set of subtrees of H_Y. The root nodes of these subtrees correspond to components in Y. We let these root nodes be the nodes of Y'. Having the external edges of each of the components corresponding to the nodes of Y' we connect such two nodes if they share an external edge. This can be done in time linear in the number of edges, assuming that the edges are labelled. The leaves in U are also leaves in Y'.

In order to consider all leaves in Y coloured $\mathcal{C}$ in Y' we let the nodes of H_Y store another vector $\mathbf{c}_{\mathcal{C}}$ and function $F_{\mathcal{C}}$. These are defined equivalently to $\mathbf{c}$ and F with the exception that they assume that all leaves in the associated component are coloured $\mathcal{C}$. These can be constructed once and for all when H_Y is constructed. We let $\mathbf{c}_{\mathcal{C}}$ and $F_{\mathcal{C}}$ be the information stored at the nodes of Y'. We note that we use $O(d^8)$ time, copying information, per node in the extraction.

6. Calculating Shared Star Quartets

The last step is the calculation of shared star quartets between T and T'. We adopt the notion of associated and compatible from butterfly quartets. We can, in the same way as above, construct polynomials G counting the number of star quartets associated with simple components of T' and compatible with the current colouring of S. As there are no star quartets associated to a leaf of T', $G(\mathbf{c^1}) = 0$. At internal nodes of T':

$$G(\mathbf{c^1}, \mathbf{c^2}, \ldots, \mathbf{c^{dc}}) = \sum_i^{d_C} \sum_{j>i}^{d_C} \sum_{k>j}^{d_C} \sum_{l>k}^{d_C} \sum_{i'}^{d} \sum_{\substack{j' \neq i'}}^{d} \sum_{\substack{k' \neq i' \\ k' \neq j'}}^{d} \sum_{\substack{l' \neq i' \\ l' \neq j' \\ l' \neq k'}}^{d} \mathbf{c}_{i'}^{\mathbf{i}} \mathbf{c}_{j'}^{\mathbf{j}} \mathbf{c}_{k'}^{\mathbf{k}} \mathbf{c}_{l'}^{\mathbf{l}}$$

The construction of G's at internal nodes of the hierarchical decomposition tree corresponds to the construction of F's at these nodes. We note that G is itself a polynomial of degree 4 with no more than d^2 variables, i.e. it can be stored and manipulated in $O(d^8)$ space and time. We conclude that we can extend both the basic and the improved algorithm, by associating G's to the nodes of the trees, into counting shared star quartets as well as the shared butterfly quartets. This enables the calculation of the quartet distance between T and T'.

References

1. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates Inc., 2004.
2. B. L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:1–13, 2001.
3. G. Estabrook, F. McMorris, and C. Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Syst. Zool.*, 34:193–200, 1985.
4. D. F. Robinson and L. R. Foulds. Comparison of weighted labelled trees. In *Combinatorial mathematics, VI (Proc. 6th Austral. Conf)*, Lecture Notes in Mathematics, pages 119–126. Springer, 1979.
5. D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.
6. M. S. Waterman and T. F. Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73:789–800, 1978.
7. D. Bryant, J. Tsang, P. E. Kearney, and M. Li. Computing the quartet distance between evolutionary trees. In *Proceedings of the 11th Annual Symposium on Discrete Algorithms (SODA)*, pages 285–286, 2000.
8. M. Steel and D. Penny. Distribution of tree comparison metrics–some new results. *Syst. Biol.*, 42(2):126–141, 1993.
9. G. S. Brodal, R. Fagerberg, and C. N.S̃. Pedersen. Computing the quartet distance between evolutionary trees in time $O(n \log n)$. *Algorithmica*, 38:377–395, 2003.
10. C. Christiansen, T. Mailund, C. N. S. Pedersen, and M. Randers. Computing the quartet distance between trees of arbitrary degree. In R. Casadio and G. Myers, editors, *WABI*, volume 3692 of *LNCS*, pages 77–88. Springer, 2005. ISBN 3-540-29008-7.

Proofs for selected lemmas for reviewers

Proof of lemma 5.1

We will show how to construct the hierarchical decomposition tree bottom up in $O(\log n)$ rounds. Before the first round each node in T is a component by itself. Consider a node v in the hierarchical decomposition tree. Let C be the associated component and let C have degree t. Let m denote the total number of simple components in C, let m_1 and m_2 the number of simple components of degree 1 and 2 respectively and $m_{3,4,\ldots,d}$ the number of simple components of degree 3 or more. As these components and the edges between them form a tree, we have $m_{3,4,\ldots,d} \leq m_1 - 2$.

The total number of edges in C is $m - 1$. Some of these correspond to illegal component compositions. Illegal compositions occur when a component of degree 3 or more is connected to a component of degree 2 or more. Let us denote these components of degree 3 or more by $C_1, C_2, \ldots, C_{m_{3,4,\ldots,d}}$. There are no more than $\sum_i^{m_{3,4,\ldots,d}} d_{C_i}$ edges corresponding to illegal compositions. In general, for trees of at least two nodes, we have $m_1 = 2 + \sum_i^{m_{3,4,\ldots,d}} d_{C_i} - 2$, this is easily seen by induction on the size of the tree. This yields $\sum_i^{m_{3,4,\ldots,d}} d_{C_i} = m_1 + 2(m_{3,4,\ldots,d} - 1)$.

Each of the remaining edges corresponds to a legal composition, so there are no less than $m - 1 - \sum_i^{m_{3,4,\ldots,d}} d_{C_i}$ of these. If $\sum_i^{m_{3,4,\ldots,d}} d_{C_i} < \frac{m}{2}$ there are at least $\frac{m}{2}$ legal compositions. In the other case, when $\sum_i^{m_{3,4,\ldots,d}} d_{C_i} \geq \frac{m}{2}$ we have $m_1 + 2(m_{3,4,\ldots,d} - 1) > \frac{m}{2}$. Using $m_{3,4,\ldots,d} \leq m_1 - 2$ as stated above, we find $m_1 + 2(m_1 - 2 - 1) > \frac{m}{2}$. This yields that $m_1 \geq \frac{m}{6}$. In either case there are at least $\frac{m}{6}$ legal compositions. A legal composition might be in conflict with another legal composition, meaning that performing the first of these results in the other becoming illegal. From Fig. 4 it is seen that no composition can be in conflict with more than $d - 1$ others. Therefore we can always compose at least $\frac{1}{6d}m$ components. Composing these yield new components inside C. Identify the number of these as m' and m_1', m_2' and $m_{3,4,\ldots,d}'$ as before. The above analysis then applies once again to these numbers. Repeating this, after k rounds, we are left with at most $c^k m$ components, where $c = (1 - \frac{1}{6d})$. Ultimately, one component remains after $\lceil \frac{-\log m}{\log c} \rceil = \lceil \log_{\frac{1}{c}} m \rceil$ rounds, meaning that the subtree rooted at v is of height $\lceil \log_{\frac{1}{c}} m \rceil \leq \frac{1}{\log \frac{1}{c}}(1 + \log m) \in O(d(1 + \log m))$[a]. Constructing the component C corresponding to v, might be done by repeatedly scanning through the components it is made up of. Each round, as it is merely a scan of the components uses time proportional to the number of components left. The total time spent is then at most of the order $m \sum_{i=0}^{d(1+\log m)} (1 - \frac{1}{6d})^i \leq 6dm$, i.e. linear in d and m. Clearly, constructing the component corresponding to the root node of the hierarchical decomposition tree then takes

---

[a]By using first order Taylor series approximations of the functions $\log x$ and $\frac{1}{x}$ we note that $\frac{1}{\log \frac{1}{c}} \in \Theta(d)$. First order Taylor series approximations $t_a(x)$ approximating $t(x)$ are of the form $t_a(x_0 + h) = t(x_0) + ht'(x_0)$ for small h. Approximating $\log x$ about the point 1 yields $\log(1 + h) \approx \log 1 + h\frac{1}{\ln 2} = h \log e$. Likewise, approximating $\frac{1}{x}$ about 1 yields $\frac{1}{1+h} \approx 1 - h$. Define $y = \frac{1}{6d}$. We see that $\frac{1}{\log \frac{1}{c}} = \frac{1}{\log \frac{1}{1-y}} \approx \frac{1}{\log 1+y} \approx \frac{1}{y \log e} = \frac{6d}{\log e}$.

12   *REFERENCES*

time $O(dn)$.

Proof of lemma 5.5

First we contract (apply a legal composition) any leaf not marked as non-contractible. We are then left with a tree with k components of degree 1. As this is a tree the number of components of degree 3 or more $m_{3,4,...,d} \leq k$. Illegal compositions are then compositions corresponding to edge contractions of edges incident to non-contractible leaves or edges incident to the $m_{3,4,...,d}$ components of degree 3 or more (as in the proof of lem. 5.1). There are no more than $\sum_i^{m_{3,4,...,d}} d_{C_i} = k + 2(m_{3,4,...,d} - 1) \leq 3k$ of such edges. This yields a decomposition consisting of at most $4k$ components.