Algorithms for Computing the Quartet Distance between Trees of Arbitrary Degree

Chris Christiansen[1], Thomas Mailund[2],
Christian N. S. Pedersen[1,2], and Martin Randers[1]

[1] Department of Computer Science, University of Aarhus,
Aabogade 34, DK-8200 Århus N, Denmark
{chrisc, cstorm, u002155}@daimi.au.dk
[2] Bioinformatics Research Center, University of Aarhus,
Høegh-Guldbergsgade 10, Bldg. 090, DK-8000 Århus C, Denmark
{mailund,cstorm}@birc.au.dk

Abstract. We present two algorithms for computing the quartet distance between trees of arbitrary degree. The quartet distance between two unrooted evolutionary trees is the number of quartets—sub-trees induced by four leaves—that differs between the trees. Previous algorithms focus on computing the quartet distance between binary trees. In this paper, we present two algorithms for computing the quartet distance between trees of arbitrary degrees. One in time $O(n^3)$ and space $O(n^2)$ and one in time $O(n^2d^2)$ and space $O(n^2)$, where n is the number of species and d is the maximal degree of the internal nodes of the trees. We experimentally compare the two algorithms and discuss possible directions for improving the running time further.

1 Introduction

The evolutionary relationship for a set of species is conveniently described by a tree in which the leaves correspond to the species, and the internal nodes correspond to speciation events. The true evolutionary tree for a set of species is rarely known, so inferring it from obtainable information is of great interest. Many different methods have been developed for this, see e.g. [8] for an overview.

Some methods infer rooted trees, where the most recent common ancestor of the set of species is represented by a root, and the direction of evolution is from the root to the leaves, while other methods infer unrooted trees, relying on an out group for inferring the direction of evolution. Most methods aim at fully resolving the evolutionary tree into a binary tree, while other methods, e.g. the Buneman [2,6] and refined Buneman [4,9], construct fully resolved binary trees only if this is well supported by the input data. Different methods often yield different inferred trees for the same set of species, and even the same method can 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.

R. Casadia and G. Myers (Eds.): WABI 2005, LNBI3692, pp. 77–88, 2005

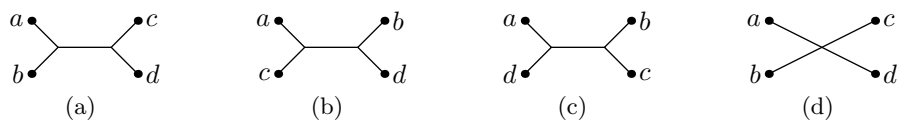


Fig. 1. The four possible quartet topologies of species a, b, c, and d. Topologies (a): $ab|cd$, (b): $ac|bd$, and (c): $ad|bc$ are denoted *butterfly* quartets, while topology (d): $\frac{a}{b} \times \frac{c}{d}$, is denoted *star* quartet.

One approach for comparing evolutionary trees is to define a distance measure between trees and compare two trees by computing this distance. Several distance measures have been proposed, e.g. the symmetric difference metric [10], the nearest-neighbour interchange metric [13], the subtree transfer distance [1], the Robinson and Foulds distance [11], and the quartet distance [7]. Each distance measure has different properties and reflects different aspects of biology.

This paper is concerned with calculating the quartet distance. 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. 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.

Previous algorithms for computing the quartet distance all focus on comparing *binary* trees and therefore avoid star quartets. Steel and Penny in [12] 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 $O(n^2)$. Brodal *et al.* in [3] present an algorithm the computes the quartet distance in time $O(n \log n)$. In this paper, we present two algorithms that computes the quartet distance between two trees of *arbitrary* degrees, i.e. trees that can contain star quartets. The first algorithm runs in time $O(n^3)$ and space $O(n^2)$, the second in time $O(n^2 d^2)$ and space $O(n^2)$, where d is the maximal degree of inner nodes in the trees.

The rest of the paper is organised as follows. In Sect. 2, we present our algorithms for computing the quartet distance between two unrooted evolutionary trees of arbitrary degree. In Sect. 3 we experimentally compare the running time of the algorithms and confirm that the actual running times concur with the theoretical results.

2 Algorithms

In this section we present three algorithms for counting the quartet distance between two non-rooted trees of arbitrary degrees over the same set of n species.

Comparing two quartets means comparing the topology of these two quartets in the two input trees. We say that a quartet is *shared* if it has the same topology in both trees. Two star quartets always have the same topology, but butterfly quartets can have three different topologies as shown in Fig. 1.

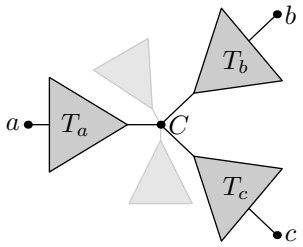


Fig. 3. The three subtrees containing the leaves a, b and c are called T_a, T_b and T_c, respectively. Any remaining trees (shown in lighter grey) are collectedly called T_{rest}.

The most obvious way to compute the quartet distance between two non-rooted trees over the same set of n species would be to explicitly compare each of the $\binom{n}{4}$ pairs of quartets. With only the input trees available, the straightforward approach for finding the topology of one quartet takes linear time, which leads to a total time usage of $O(n^5)$.

We first present an algorithm that runs in time $O(n^4)$ and space $O(n)$, which we modify to run in time $O(n^3)$ but space $O(n^2)$. Then we establish some terminology and present a third algoritm, using $O(n^2d^2)$ time and $O(n^2)$ space.

2.1 General Quartet Distance in Time $O(n^4)$ and Space $O(n)$

Given two input trees T and T' over the same set of n species, we can compute the quartet distance in time $O(n^4)$, if the quartet topology for each set can be found in constant time in both input trees. We will show how to achieve this by focusing on the *centers* between triplets of leaves: Given leaves a, b, and c, there is a unique inner node C, the center of a, b and c, in which the paths from a to b, a to c and b to c are joined, see Fig. 2.

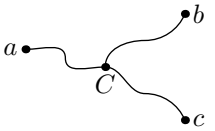


Fig. 2. The *center*, C, of a, b and c.

Given leaves $a, b, c \in T$ with center C, let the subtree containing a be denoted T_a, similarly for leaves b and c. Any remaining subtrees of C are collectedly denoted T_{rest}, see Fig. 3. For each leaf x different from a, b and c, a quartet is defined, and its topology in T can be easily determined from the center: if $x \in T_a$ then the topology of the quartet is $ax|bc$, and if $x \in T_b$, the quartet is $bx|ac$ and if $x \in T_c$ the quartet is $ab|cx$. If $x \in T_{\text{rest}}$, then the topology is $\begin{smallmatrix} a & c \\ b & x \end{smallmatrix}$. Similarly, the quartet topologies for a, b, c and x can be determined in T' given the center of a, b, and c in T'.

To determine the quartet distance between two trees, each of the $\binom{n}{3} \in O(n^3)$ triplets of leaves is processed sequentially in both trees: The centers can be found, and the leaves in the different subtrees accumulated, in linear time. Assuming leaves are numbered $0, \ldots, n-1$, the topology of each of the $n-3$ quartets containing a, b and c can be stored in an array for each tree, where entry i contains the topology of the quartet a, b, c and the ith leaf. In this way, the

topology of the quartets can be compared directly. Each of the $O(n^3)$ triplets can be processed in linear time, making the algorithm run in time $O(n^4)$ and space $O(n)$. Since each quartet is processed four times—once for each triplet it contains, the number of quartets that have different topologies must be divided by four to get the correct quartet distance.

The algorithm uses no data structures other than the trees and arrays, little memory and no complex methods, so it is very easy to implement. It is also very easy to extend it to count the number of *shared* quartet topologies between k trees over the same set of n species in time $O(k \cdot n^4)$ and space $O(k \cdot n)$.

2.2 General Quartet Distance in Time $O(n^3)$ and Space $O(n^2)$

Instead of counting quartets with different topologies in the input trees, we now count the number of shared quartets and then subtract this number from the total number of quartets, $\binom{n}{4}$, to get the quartet distance.

Given rooted subtrees T_x and T'_x of T and T' respectively, we can pre-compute the size of the intersection of leaf sets, denoted $|T_x \cap T'_x|$. There are $O(n^2)$ pairs of such subtrees, and all intersection sizes can be computed and stored in time and space $O(n^2)$, see [5] for details. Using these precomputed sizes, we can improve the running time of the above algorithm to $O(n^3)$.

Consider leaves a, b and c. Let centers and subtrees in T and T' be defined as above. Use prime ($'$) to denote the center and subtrees in T', and no prime for the center and subtrees in T. Any leaf, x, in T_a gives a butterfly quartet of the form $ax|bc$ in T, and any leaf x' in T'_a gives a butterfly quartet of the form $ax'|bc$ in T'. Therefore, any leaf x, $x \neq a$, in both T_a and T'_a, represents a shared butterfly quartet in the two trees. The same applies for b and c. The number of shared butterfly quartets containing a, b and c can therefore be computed by the expression:

$$|T_a \cap T'_a| + |T_b \cap T'_b| + |T_c \cap T'_c| - 3\,.$$

This number can be computed in constant time, since the sizes of the intersections are precomputed. Any leaf x in T_{rest} gives a star quartet in T, and any leaf x' in T'_{rest} gives a star quartet in T'. It follows that the number of shared star quartets containing a, b and c can be computed by the expression:

$$|T_{\text{rest}} \cap T'_{\text{rest}}|\,.$$

Since T_{rest} and T'_{rest} potentially consist of a large number of subtrees, it is not clear how to compute $|T_{\text{rest}} \cap T'_{\text{rest}}|$ in constant time. However, $|T_{\text{rest}} \cap T'_{\text{rest}}|$ can be expressed in terms of T_a, T_b, T_c, T'_a, T'_b and T'_c. Any leaf in T'_{rest} is either in T_a, T_b, T_c or T_{rest}, and never in two of those at the same time, since they are disjoint. Thus:

$$|T_{\text{rest}} \cap T'_{\text{rest}}| = |T'_{\text{rest}}| - (|T'_{\text{rest}} \cap T_a| + |T'_{\text{rest}} \cap T_b| + |T'_{\text{rest}} \cap T_c|)\,.$$

Any element in T_a that is not in T'_a, T'_b or T'_c is in T'_{rest}. The same applies for elements in T_b and T_c. This gives the equations:

$$|T_a \cap T'_{\text{rest}}| = |T_a| - (|T_a \cap T'_a| + |T_a \cap T'_b| + |T_a \cap T'_c|)\,,$$
$$|T_b \cap T'_{\text{rest}}| = |T_b| - (|T_b \cap T'_a| + |T_b \cap T'_b| + |T_b \cap T'_c|)\,,$$
$$|T_c \cap T'_{\text{rest}}| = |T_c| - (|T_c \cap T'_a| + |T_c \cap T'_b| + |T_c \cap T'_c|)\,.$$

Since there are n leaves in the trees, it follows directly that:

$$|T'_{\text{rest}}| = n - (|T'_a| + |T'_b| + |T'_c|)\,.$$

Combining all these equations give an expression for $|T_{\text{rest}} \cap T'_{\text{rest}}|$ that can be computed in constant time. So given a center of three leaves in each tree, the number of shared topologies of quartets containing these leaves can be found in constant time. Assuming the centers can be found in constant time, the algorithm will run in time $O(n^3)$, since there are $O(n^3)$ different triplets of leaves.

Finding the centers in constant time is done by finding a linear number of centers in linear time: For each pair of leaves, a and b, the path from a to b can be found in linear time by a single traversal of the tree. Each inner node in the path is the center of all triplets a, b, c where c can be reached from the node via an edge not in the the path, see Fig. 4 for an illustration.

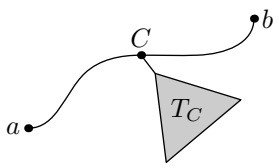


Fig. 4. For node C on the path from a to b, any subtree T_C, rooted in C and not on the path, defines a set of leaves, c, for which C is the center of a, b, and c.

Assuming leaves are numbered $0, \ldots, n-1$, the center of each of the $n-2$ triples containing a and b can be put in an array, where entry i contains the center of a, b and the ith leaf. All internal nodes on the path from a to b is a covering and disjoint set of centers for all triplets containing a and b. Therefore, filling all array entries can be done by a single traversal of the tree. Using these arrays, and the equations above, the number of shared quartets containing a fixed pair of leaves can be found in linear time. Since there are $O(n^2)$ pairs of leaves, the algorithm runs in time $O(n^3)$. The space consumption is $O(n^2)$, since all the intersection sizes of induced subtrees must be stored. As before, all quartets are counted four times, so the final count must be divided by four.

2.3 General Quartet Distance in Time $O(n^2d^2)$ and Space $O(n^2)$

Given four leaves and their quartet topology in T and T', there are four possible cases: **1**. The quartet has a star topology in both trees (shared quartet). **2**. The quartet has an equal butterfly topology in both trees (shared quartet). **3**. The quartet has a different butterfly topology in the trees. **4**. The quartet has a butterfly topology in one tree, and a star topology in the other tree. For $i = 1, 2, 3, 4$, let Q_i be the number of quartets in case i above. The quartet distance $\text{qdist}(T, T')$ between T and T' is $Q_3 + Q_4$. Let BQ and BQ' be the

number quartets that have butterfly topologies in T and T', respectively. The main observation of our approach is that $Q_4 = BQ + BQ' - 2(Q_2 + Q_3)$, which gives the following expression:

$$\text{qdist}(T, T') = Q_3 + Q_4 = BQ + BQ' - 2Q_2 - Q_3$$

Surprisingly, this means that the general quartet distance can be calculated without directly considering the star quartets. What we need are three algorithms: one that can count the number of quartets with butterfly topology in a single general tree (for computing BQ and BQ'), one that can count the number of quartets that share the same butterfly topology in two general trees (for computing Q_2), and one that can count the number of quartets that have different butterfly topologies in two general trees (for computing Q_3).

The first algorithm can be implemented using the second algorithm: counting the number of quartets with the same butterfly topology in two equal trees corresponds to counting the total number of butterfly quartets in that tree. Therefore we only need to consider the second and third algorithm. Both can be made by extending the algorithm described in [5].

The algorithm for counting butterfly quartets with the same topology uses the concepts of *directed edges*, *directed quartets*, and *claims*, that we define below:

A butterfly quartet $ab|cd$ is *induced* by edges which separate a, b from c, d, note that the inducing edge implies the topology of the quartet. Since several edges may induce the same butterfly quartet, it is convenient to look at directed edges: every edge e defines two rooted subtrees T_1 and T_2; instead of viewing e as a single undirected edge, we can view it as two directed edges e_1 and e_2, and we say that T_1 is *in front* of e_1 and T_2 is *behind* e_1 and similarly for e_2: T_2 is in front of e_2 and T_1 is behind e_2, see Fig. 5.

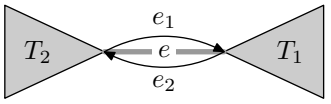


Fig. 5. Edge e defines rooted trees T_1 and T_2 and directed edges e_1 and e_2.

A butterfly quartet $ab|cd$ defines two *directed* quartets: $ab \rightarrow cd$ and $ab \leftarrow cd$; for undirected edge e, inducing quartet $ab|cd$, the corresponding directed edges e_1 and e_2 induces the directed quartets $ab \rightarrow cd$ and $ab \leftarrow cd$.

To each directed quartet, $ab \rightarrow cd$, we can uniquely associate a directed edge, e_1 such that a and b are leaves in the tree *behind* e_1, and such that c and d are leaves in *different* subtrees of the root of the tree *in front* of e_1, see Fig. 6. We call such a tree substructure a *claim*, written $T_1 \xrightarrow{e_1} (T_2, T_3)$, and say that the edge e_1 *claims* the directed quartet $ab \rightarrow cd$ and we also say that an edge e_1 claims an undirected quartet $ab|cd$ if it claims one of its directed quartets.

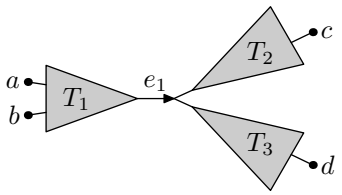


Fig. 6. The directed edge e_1 claims all directed quartets $ab \rightarrow cd$ where $a, b \in T_1$, $c \in T_2$ and $d \in T_3$.

Since each (undirected) butterfly quartet defines exactly two directed quartets, and each directed quartet is claimed by exactly one directed edge, each butterfly quartet is claimed by exactly two directed edges.

Now, Consider a pair of claims, $cl = T_1 \xrightarrow{e} (T_2, T_3)$ and $cl' = T'_1 \xrightarrow{e'} (T'_2, T'_3)$ in T and T', respectively. The number of oriented butterfly quartets shared by the claims can be computed by the expression:

$$\text{count}(cl, cl') = \binom{|T_1 \cap T'_1|}{2} \cdot (|T_2 \cap T'_2| \cdot |T_3 \cap T'_3| + |T_2 \cap T'_3| \cdot |T_3 \cap T'_2|) \quad (1)$$

and the total number of shared oriented butterfly quartets between T and T' is given by:

$$2Q_2 = \sum_{cl \in T} \sum_{cl' \in T'} \text{count}(cl, cl') \quad (2)$$

summing over all pairs of claims from the two trees.

Since each butterfly quartet corresponds to exactly two oriented butterfly quartets, the number of shared butterfly quartets is obtained by dividing the sum with two. Since the size of the intersection of subtrees is precomputed, $\text{count}(cl, cl')$ can be computed in constant time, and the sum can thus be computed in time proportional to the number of claims in T times the number of claims in T'.

For binary trees, there is a $1-1$ correspondence between claims and directed edges pointing to internal nodes, and therefore $O(n)$ claims per tree, resulting in a $O(n^2)$ time algorithm. For general trees, a directed edge e pointing to a node of degree d will be part of $\binom{d-1}{2}$ claims, see Fig. 7. Thus, the straightforward application of the sum in (2) result in an $O(n^2d^4)$ time algorithm for $d-$airy trees, since each edge in each tree can lead to $O(d^2)$ claims.

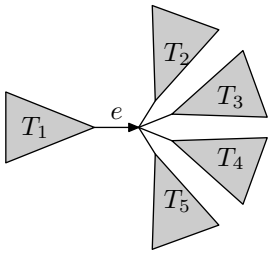
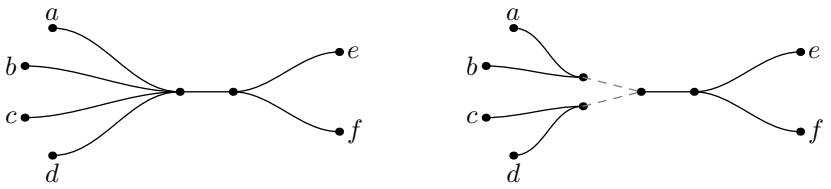


Fig. 7. The edge e is not part of a unique claim, but part of all claims $T_1 \xrightarrow{e} (T_i, T_j)$ for $i, j = 2, 3, 4, 5, i \neq j$.

The reduction in time to $O(n^2d^2)$ is achieved by a transformation of the input trees T and T' into binary trees, annotated with information about the original trees, from which the butterfly quartets shared in the original trees can be calculated.

Expanding two trees of arbitrary degree is done by expanding every node with degree higher than three to a number of binary nodes. This adds a linear number of edges and nodes to the trees, so it does not change their sizes asymptotically. The two expanded trees induce all the butterfly quartets in the two original non-expanded input trees, but they also induce additional butterfly quartets due to the newly added nodes and edges: each star quartet in the original trees will be translated into butterfly quartets by the introduction of new edges, when resolving high-degree nodes, see Fig. 8. Thus, summing the contribution of all pairs of claims using (2) on the expanded trees will count too many butterfly quartets.



(a) Original tree with high-degree node.

(b) Expanded tree, new edges shown dashed in grey.

Fig. 8. A tree with a high-degree node (a), and an expanded tree for this tree (b). The star quartets in the tree on the left becomes butterfly quartets on the right, due to the newly introduced edges, e.g. $\begin{smallmatrix} a & c \\ b & e \end{smallmatrix}$ becomes $ab|ce$.

Consider an edge, e_1, present in the original tree, and any edge e_2 resulting from the expansion of the tree, where e_2 is reachable from e_1 through newly introduced edges exclusively, see Fig. 9. These two edges, *together*, claim a set of butterfly quartets found in both the extended and the original tree: for leaves $a, b \in T_1$ behind e_1 and $c \in T_2$ and $d \in T_3$ in front of e_2 have, by construction, quartet topology $ab|cd$ in the extended tree, but also in the original tree since e_1 separates a and b from c and d in the original tree.

Formally, we let $\dashrightarrow^*$ denote the reflexive and transitive closure of newly introduced edges, that is, for $e_1 \dashrightarrow^* e_2$, e_2 is either e_1 or e_2 is reachable from e_1 following only newly introduced edges. If e_2 form a claim in the extended tree, we say that $e_1 \dashrightarrow^* e_2$ is an *extended claim*. For extended claims $ecl \in T$ and $ecl' \in T'$ we can calculate the number of shared quartets using (1), but with the sub-trees from the extended tree, defined by the extended claims.

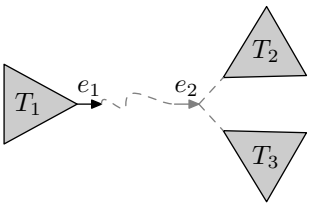


Fig. 9. An extended claim from original edge e_1 through newly introduced edge e_2.

The following propositions establishes that we can count the ordered butterfly quartets shared in the original trees using the extended claims in the extended trees, ET and ET', which gives us.

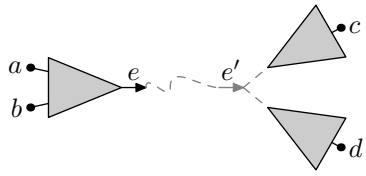
$$2Q_2 = \sum_{ecl \in ET} \sum_{ecl' \in ET'} \text{count}(ecl, ecl') \tag{3}$$

Proposition 1. *Each directed butterfly quartet, $ab \to cd$ in the original tree corresponds to exactly one extended claim in the extended tree.*

Proof. Consider the quartet $ab \to cd$ in the original tree, and corresponding claim $T_{ab} \xrightarrow{e} (T_c, T_d)$ in the original tree. The node separating trees T_{ab}, T_c, and T_d is either a binary node (degree 3) or a node of higher degree.

In the first case, where the node has degree 3, it will not have been expanded in ET and the claim will also be an extended claim in ET, and the only extended claim in ET that claims $ab \to cd$.

In the second case, where the node has degree higher than 3, it will be expanded into a sub-tree of newly introduced nodes and edges. Among the newly introduced nodes, there is a node that separates the subtrees containing c and d, since they were separated by the high degree node that was expanded. Let e' denote the edge pointing to this node, not from the trees containing c and d; the situation is then as this:



This is an extended claim, claiming the oriented quartet $ab \rightarrow cd$ in ET. Any other extended claim, claiming $ab \rightarrow cd$, must contain e', by definition, and an original edge e'' such that a and b are found in the tree behind e'' and such that $e'' \dashrightarrow^* e'$. Assume that such an e'' exist, and that $e'' \neq e$. Since e separates a and b from the expanded node that originally separated ab from c and d, any edge separating ab from cd must either be on the path from e to e', or be behind e. Assume e'' is on the path. Since $e \dashrightarrow^* e'$ the path only contain new edges, which contradicts that e'' is an original edge. Assume that e'' is behind e, which means that e is on the path from e'' to e'. Since $e'' \dashrightarrow^* e'$ by assumption, this path consist of new edges, which contradicts that e is an original edge. The conclusion is the edge e'' cannot exist, thus the extended claim is unique. □

Proposition 2. *All directed quartets $ab \rightarrow cd$ claimed by extended claims in the extended tree correspond to directed quartets in the original tree.*

Proof. If $ab \rightarrow cd$ is claimed by an extended claim, through original edge e_1 and new edge e_2, in ET, then, by the construction of ET, the original edge e_1 separates ab from cd in T. Thus e_1 induces, but does not necessarily claim, the quartet $ab \rightarrow cd$ in T. □

Using precomputed sizes of intersecting subtrees, as before, we can calculate $\text{count}(ecl, ecl')$ in constant time, so to achieve the desired complexity for computing (3) we need to shown that we can calculate all pairs of extended claims in time $O(n^2d^2)$; we do this by showing how we can calculate all extended claims for a single tree in time $O(nd)$.

When building the extended tree, we can tag each edge with a flag specifying if it is an edge in the original tree or if it is a newly introduced edge. With these tags, we can iterate through all original edges in the extended tree by a simple tree traversal in linear time. For each original edge, e_1, we can find all e_2 such that $e_1 \dashrightarrow^* e_2$ by a simple traversal starting at e_1 following only newly introduced edges. Since the newly introduced edges reachable from e_1 found in this way are all edges in a tree resulting from a node of degree at most d, this search can be done in time $O(d)$ giving a total time for finding all extended claims in a single tree of $O(nd)$.

For counting butterfly quartets with different topologies we use an algorithm that works almost in the same way as the one just described, but it uses an alternative version of (1), dcount. dcount counts the number of butterfly quartets that have different topologies instead of the ones with the same topology. Similar to (1), dcount can be calculated in constant time given two claims or extended claims.

Observe that when comparing two extended claims ecl and ecl', butterfly quartets associated to these claims, that have different topologies, are those quartets where exactly one leaf is in $T_1 \cap T_1'$. The number of such quartets can be counted using the following expression:

$$\begin{aligned}\mathrm{dcount}(ecl, ecl') = \; & |T_1 \cap T_1'| \cdot |T_1 \cap T_3'| \cdot |T_2 \cap T_2'| \cdot |T_3 \cap T_1'| + \\ & |T_1 \cap T_1'| \cdot |T_1 \cap T_2'| \cdot |T_2 \cap T_3'| \cdot |T_3 \cap T_1'| + \\ & |T_1 \cap T_1'| \cdot |T_1 \cap T_2'| \cdot |T_2 \cap T_1'| \cdot |T_3 \cap T_3'| + \\ & |T_1 \cap T_1'| \cdot |T_1 \cap T_3'| \cdot |T_2 \cap T_1'| \cdot |T_3 \cap T_2'|\end{aligned}$$

Counting the total number of butterfly quartets with different topologies in the two trees is done by comparing all pairs of extended claims.

$$4Q_3 = \sum_{ecl \in ET} \sum_{ecl' \in ET'} \mathrm{dcount}(ecl, ecl') \tag{4}$$

Since each undirected butterfly quartet is claimed twice in each tree, the ones with different topologies are counted four times—once for each combination of claims. Therefore, the number of butterfly quartets with different topology is obtained by dividing the sum with four.

3 Experiments

We have implemented the algorithms[1] and done a set of experiments to validate the theoretical time complexities of our algorithms. The performance of the $O(n^2d^2)$ algorithm has been verified by running the algorithm on trees where all nodes (except perhaps one, with a smaller degree) have the same degree d. We have run the algorithm for various degrees d but fixed numbers of leaves n. Similarly we have kept d constant while changing n, to verify the running time with respect to n. The last experiment has been done for both algorithms.

Fig. 10 shows the running time of the algorithm as a function of d. For each of n, the running time is $O(d^2)$, which verifies the theoretical time complexity with respect to d.

The running times shown in Fig. 11 is plotted as a function of the number of leaves n. For each d the running time as $O(n^2)$ and $O(n^3)$ respectively. Again this agrees with the theoretical time complexity. As expected the $O(n^3)$ does not seem to be dependent on d, however for $d = 3$ the algorithm is slightly slower.

[1] The algorithms are implemented in Java and can be obtained by contacting one of the authors.

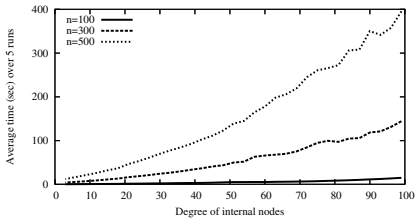
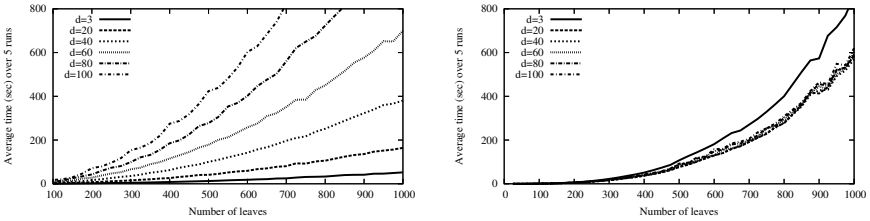


Fig. 10. Running time of the $O(n^2d^2)$ algorithm as a function of the degree d of the internal nodes for three fixed numbers of leaves n.



(a) $O(n^2d^2)$ algorithm. (b) $O(n^3)$ algorithm.

Fig. 11. Running time of the $O(n^2d^2)$ and $O(n^3)$ algorithms as a function of the number of leaves n for six fixed internal node degrees d. For every d it is verified that the running time as a function of n is $O(n^2)$ and $O(n^3)$ respectively.

This can be explained by the longer paths between every pair of leaves in such trees, since the algorithm is based on processing paths between pairs of leaves.

The two graphs also show that the choice of algorithm for computing the quartet distance should depend on both n and d as expected. However note that all trees in our testing scenario have many internal nodes with a relatively high degree. Trees generated using Buneman [2, 6] and refined Buneman [4, 9] can be expected to have only a few internal nodes of high degree if enough data is available. On such trees the $O(n^2d^2)$ time algorithm can be expected to run faster than in our tests, since only a few internal nodes have to be expanded. One can view the graphs in Fig. 11 as a worst case guideline of when to use the $O(n^3)$ algorithm. A direct comparison of the running times can be seen in Fig. 12. Since Fig. 11(b) shows no significant variance in the running time for $d \geq 20$, we choose only to show two different plots for the $O(n^3)$ algorithm.

Our experiments show that the algorithms described in this paper perform within the theoretical time bounds. More experiments are needed to investigate the performance of the algorithms on actual experimental data, e.g. trees generated by the Buneman methods.

Further work will be directed at improving the time complexity of computing the quartet distance on trees of arbitrary degree. The use of leaf intersection sizes in our algorithms prevents them from having a running time faster than $O(n^2)$. Consequently work on improving the $O(n^2d^2)$ time algorithm should be directed

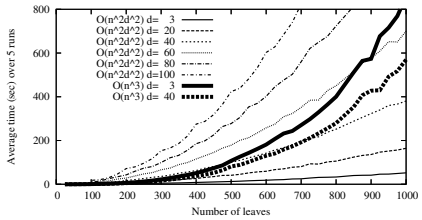


Fig. 12. Comparison of the running times.

at removing one or both factors of d. Another approach to reducing the time complexity is to adapt the ideas from the $O(n \cdot \log n)$ time algorithm in [3] to trees of arbitrary degree. Preliminary studies show that straight forward adaptations cannot preserve the $O(n \log n)$ time complexity. Even if this is possible there might still be need for our algorithm, since it has very little overhead to the $O(n \log n)$ time algorithm.

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. V. Berry and D. Bryant. Faster reliable phylogenetic analysis. In *Proc. 3rd International Conference on Computational Molecular Biology (RECOMB)*, 1999.
3. G. S. Brodal, R. Fagerberg, and C. N. S. Pedersen. Computing the quartet distance between evolutionary trees in time $O(n \log n)$. *Algorithmica*, 38:377–395, 2003.
4. D. Bryant and V. Moulton. A polynomial time algorithm for constructing the refined buneman tree. *Applied Mathematics Letters*, 12:51–56, 1999.
5. 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.
6. P. Buneman. The recovery of trees from measures of dissimilarity. In F. Hodson, D. Kendall, and P. Tautu, editors, *Mathematics in Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, 1971.
7. 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.
8. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates Inc., 2004.
9. V. Moulton and M. Steel. Retractions of finite distance functions onto tree metrics. *Discrete Applied Mathematics*, 91:215–233, 1999.
10. 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.
11. D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.
12. M. Steel and D. Penny. Distribution of tree comparison metrics–some new results. *Syst. Biol.*, 42(2):126–141, 1993.
13. M. S. Waterman and T. F. Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73:789–800, 1978.