Algorithms for Molecular Biology BioMed Central

Research



# Fast calculation of the quartet distance between trees of arbitrary degrees

Chris Christiansen[1], Thomas Mailund*[2], Christian NS Pedersen*[1,3], Martin Randers[1] and Martin Stig Stissing[1]

Address: [1]Department of Computer Science, University of Aarhus, Aabogade 34, DK-8200 Århus N, Denmark, [2]Department of Statistics, University of Oxford, 1 South Parks Road Oxford OX1 3TG, UK and [3]Bioinformatics Research Center, University of Aarhus, Høegh-Guldbergsgade 10, Bldg. 090, DK-8000 Århus C, Denmark

Email: Chris Christiansen - chrisc@daimi.au.dk; Thomas Mailund* - mailund@stats.ox.ac.uk; Christian NS Pedersen* - cstorm@birc.au.dk; Martin Randers - martin.randers@daimi.au.dk; Martin Stig Stissing - stissing@daimi.au.dk

* Corresponding authors

Published: 25 September 2006

*Algorithms for Molecular Biology* 2006, **1**:16 doi:10.1186/1748-7188-1-16

Received: 18 May 2006
Accepted: 25 September 2006

This article is available from: http://www.almob.org/content/1/1/16



## Abstract

**Background:** A number of algorithms have been developed for calculating the quartet distance between two evolutionary trees on the same set of species. The quartet distance is the number of quartets – sub-trees induced by four leaves – that differs between the trees. Mostly, these algorithms are restricted to work on binary trees, but recently we have developed algorithms that work on trees of arbitrary degree.

**Results:** We present a fast algorithm for computing the quartet distance between trees of arbitrary degree. Given input trees $T$ and $T'$, the algorithm runs in time $O(n + |V|\cdot|V'| \min\{id, id'\})$ and space $O(n + |V|\cdot|V'|)$, where $n$ is the number of leaves in the two trees, $V$ and $V$ are the non-leaf nodes in $T$ and $T'$, respectively, and $id$ and $id'$ are the maximal number of non-leaf nodes adjacent to a non-leaf node in $T$ and $T'$, respectively. The fastest algorithms previously published for arbitrary degree trees run in $O(n^3)$ (independent of the degree of the tree) and $O(|V|\cdot|V'|\cdot id\cdot id')$, respectively. We experimentally compare the algorithm with existing algorithms for computing the quartet distance for general trees.

**Conclusion:** We present a new algorithm for computing the quartet distance between two trees of arbitrary degree. The new algorithm improves the asymptotic running time for computing the quartet distance, compared to previous methods, and experimental results indicate that the new method also performs significantly better in practice.

## Background

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. [1] for an overview.

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

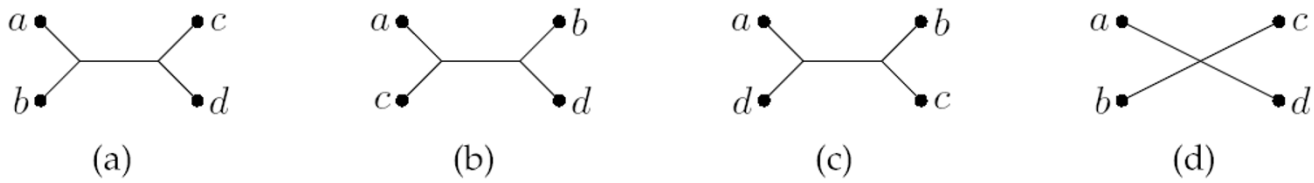


**Figure 1**

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

species. 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 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 [2], the nearest-neighbour interchange metric [3], the subtree transfer distance [4], the Robinson and Foulds distance [5], and the quartet distance [6]. Each distance measure has different properties and reflects different aspects of biology.

This paper is concerned with calculating the quartet distance. A *quartet* is a set of four species, the *quartet topology* induced by an evolutionary tree 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.

Steel and Penny [7] pointed at Doucettes unpublished work [8] which presented an algorithm for computing the quartet distance in time $O(n^3)$, where $n$ is the number of species. Bryant et al. in [9] presented an improved algorithm which computes the quartet distance in time $O(n^2)$ for binary trees. Brodal *et al*. in [10] showed how to compute the quartet distance in time $O(n \log n)$ considering binary trees. For arbitrary degree trees, the quartet distance can be calculated in time $O(n^3)$ or $O(n^2d^2)$, where $d$ is the maximum degree of any node in any of the two trees, as shown by Christiansen *et al*. [11].

## Results and discussion

In [11], we presented an algorithm for computing the quartet distance between trees of arbitrary degree. It runs in time $O(n^2d^2)$ and space $O(n^2)$, where $n$ is the number of leaves in each tree and $d$ is the maximal degree found in either of the trees. In this paper, we present an improved algorithm running in time $O(n + |V||V'| \min\{id, id'\})$ and space $O(n + |V||V'|)$, where $|V|$ and $|V'|$ are the number of internal (non-leaf) nodes in the two input trees, and $id$ and $id'$ are the maximal degree of an internal node, when disregarding edges to leaves, in the two trees.

### *Time analysis for different types of trees*

The terms $|V|$, $id$, $|V'|$ and $id'$ are all clearly $O(n)$, but on the other hand neither $|V|$ and $id$ nor $|V'|$ and $id'$ are independent. Intuitively, if there are a lot of internal nodes in a tree, they will not have a very large internal degree. We address in this section, how this dependency will affect the running time for different types on trees.

The worst theoretical running time of the algorithm for calculating the quartet distance presented above is $O(n^3)$. Consider a tree with an internal node of degree $\frac{n}{2}$, connected to $\frac{n}{2}$ internal nodes of degree three each connected to two leaves, see Fig. 2. Such a tree has $n$ leaves, $O(n)$ internal nodes and a maximal internal degree that is $O(n)$. If the algorithm is run on two such trees, the running time will be $O(n^3)$. In $d$-ary trees (trees where all internal nodes have degree $d$) $|V| = O\left(\frac{n}{d}\right)$, the time complexity of calculating the quartet distance will be $O\left(\frac{n^2}{d}\right)$.

The two cases above are somewhat extreme. The first case has a very large gap between the maximal and minimal degree of internal nodes, while the second has little or no gap. The theoretical performance of the algorithm on the two types of trees reflects this difference. Let $d_{min} = \min\{\min_v d_v, \min_{v'} d_{v'}\}$, be the minimal degree of any

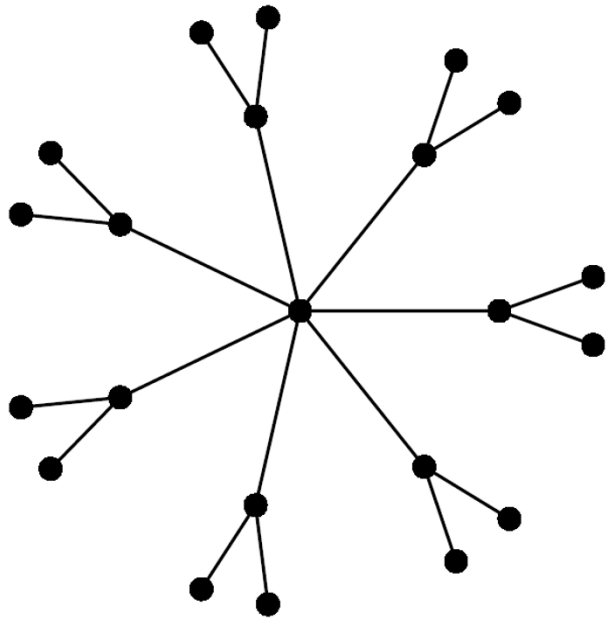

**Figure 2**

**A worst case input tree for the algorithm**. A tree with an internal node of degree $\frac{n}{2}$, connected to $\frac{n}{2}$ internal nodes of degree three each connected to two leaves. This tree has both a maximal degree of $O(n)$ and at the same time $O(n)$ inner nodes.

internal node in either tree, then each tree has $O(\frac{n}{d_{min}})$ internal nodes and the time complexity is $O(\frac{n^2}{d_{min}^2}\min\{id, id'\})$. If $\min\{id, id'\}$ is $O(d_{min}^2)$ the time usage of calculating the quartet distance will be $O(n^2)$. In the following section we will do practical verification of the theoretical results in this section.

#### *Experimental running times*

The graphs in Fig. 3 show the running time for comparing worst case trees (see Fig. 2), ($d$-ary trees and random trees. There are six types of ($d$-ary trees; binary, 6-ary, 15-ary, and 30-ary and two types of random trees; r8s-based (see [12]) and trees with random topologies. The trees generated by r8s are binary, but by contracting edges, we can get trees of arbitrary degree (contracting an edge $e$ connecting nodes $u$ and $v$ means removing $u$ and $e$ and attaching the rest of $u$'s edges to $v$). Each edge is contracted with a probability that is inversely proportional with its length, i.e. a short edge has a higher probability of being contracted than a long edge. The trees with random topology are generated by adding leaves one by one, starting with a tree of size 2. A leaf can be added by attaching it to a random inner node or by spliting a random edge with a new node, to which the leaf is attached.

The running time for worst case input trees (as described in the previous section) is $O(n^3)$, because such trees have $O(n)$ internal nodes and $\min\{id, id'\}$ is $O(n)$. This is supported by the first graph in Fig. 3, which shows that the plot of the polynomial $n^3$ (representing the best sum-of-squares fit of the polynomial $c \cdot n^3$ to the data-points) is closest to the plot of the running times with regard to slope.

The running time on the algorithm on $d$-ary trees is $O(\frac{n^2}{d})$. The plots of the running times in the second graph are parallel, and one of them is plotted directly on top of a plot of the polynomial $n^2$ (here $c \cdot n^2$ is fitted to the data-points for each $d$ separately; the different colors match the $d$ colors). This supports that they all have a running time of $O(n^2)$ for fixed $d$'s. The graph also shows that higher degrees give lower running times, which is also expected. The reason why the algorithm is more than twice as fast on 6-ary trees than it is on binary trees, is that the number of internal nodes in 6-ary trees is less than in binary trees, and even though $|V|$ is $O(n)$ in both cases, that difference has an impact on the running time. The last graph shows the running time of the algorithm on trees created as either random trees (each topology is equally likely) or trees simulated using r8s (with edge contraction as described above). We have no theoretical running time for this data, but the graphs show that the running time is $O(n^2)$. Even though the plotted data is only a small random sample, this indicates that many pairs of trees actually have the property that $\min\{id, id'\}$ is $O(d_{min}^2)$. Therefore it is not unreasonable to expect that our algorithm runs in time $O(n^2)$ on trees used in practice. All experiments were performed on a standard PC (Pentium 4, 3 GHz, 1 Gb Ram) running Linux Fedora Core 3.

#### *Comparison with existing algorithms*

In Fig. 4 we compare the running time of the new algorithm with the $O(n^2d^2)$ and $O(n^3)$ time algorithms from [11] on random and r8s simulated trees. In Fig. 5 we compare the running time of the new algorithm with the other two algorithms on Buneman and refined Buneman trees built for a range of Pfam [13] derived distance matrices using the tool in [14]. Buneman and refined Buneman trees are not binary unless this is well supported by the input distance matrix, and thus represent the kind of trees

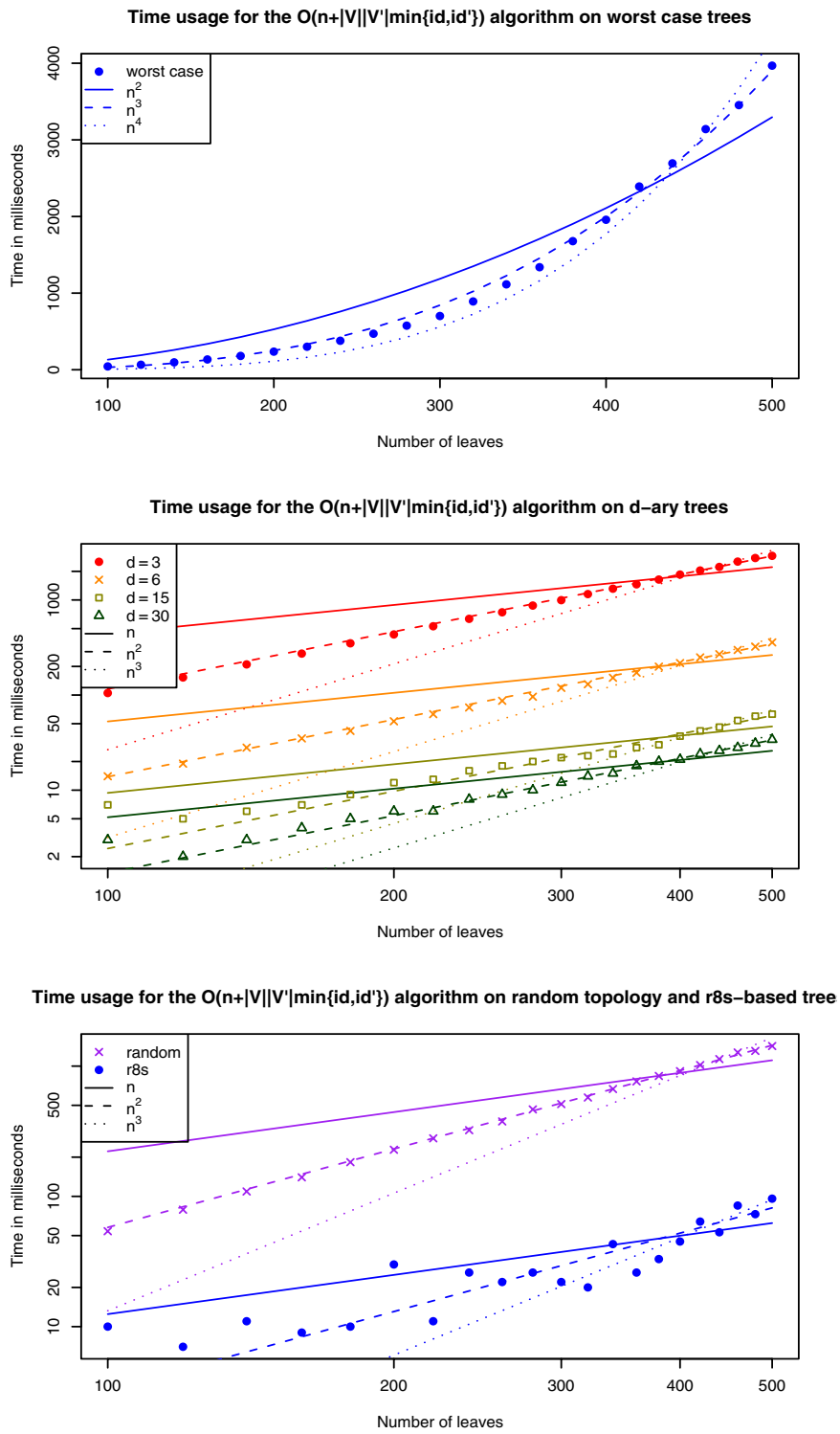


**Figure 3**

**Experimental running times**. The running time of the algorithm for worst case trees, *d*-ary trees and random trees. The lines plots the polynomials $c.\ n^i$, where $c$ is a fitted constant and $i \in [1, 4]$. The two bottommost plots are in log-scale on both the x- and y-axis.

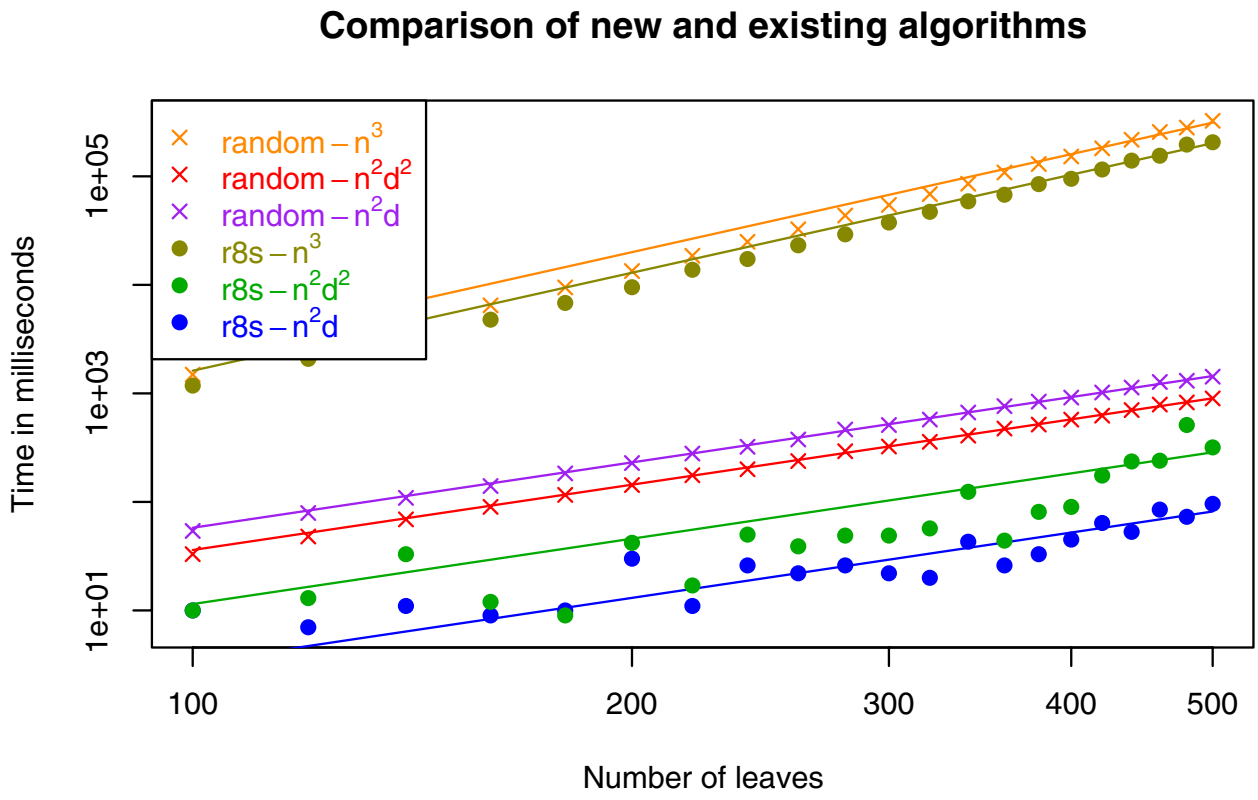


**Figure 4**
**Comparisons with earlier algorithms on random and r8s trees**. The running time for the new algorithm compared to the existing $O(n^2d^2)$ and $O(n^3)$ time algorithms for random and r8s trees. The lines are fitted polynomial *c*. $n^2$, for the case of the new algorithm (denoted $n^2d$ in the legend) and the $O(n^2d^2)$ algorithm, and the polynomial *c*. $n^3$ for the $O(n^3)$ algorithm. The plot is in log-scale on both the x- and y-axis.

which can only be compared by methods which allow for trees of arbitrary degrees. In both experiments, the $O(n^3)$ time algorithm is slowest by a large margin for all plotted sizes of $n$. The new algorithm is consistently faster than the $O(n^2d^2)$ time algorithm for the r8s (with edge contraction) simulated trees and for the Buneman and refined Buneman trees. For random trees the previous $O(n^2d^2)$ time algorithm is slightly faster in practice. This difference is most likely caused by the additional overhead of precomputing the sums used by the new $O(n^2d)$ time algorithm compared to the previous $O(n^2d^2)$ time algorithm in order to improve the asymptotic worst case running time (see method section). For trees of low degree, the overhead might dominate the factor $d$ by which the worst case running time of the new algorithm is improved. The observed running times on random trees thus indicate that over selection of random trees consists of trees of low degree, whereas the r8s simulated, Buneman, and refined Buneman trees are trees with a few nodes of high degree which more than compensate for the additional overhead of dealing with nodes of low degree. In conclusion, we find that the experimental comparison of the new algorithm with the previously developed algorithms indicate that the new algorithm not only improves on the theoretical asymptotic running time, but also improves the running time in practice if the input trees contain a few nodes of high degree.

## Conclusion

We have constructed an algorithm for finding the quartet distance between two trees of arbitrary degree. It runs in time $O(n + |V||V'| \min\{id, id'\})$ and uses space $O(n + |V||V'|)$, where $n$ is the number of leaves in the trees, $|V|$ and $|V'|$ are the number of internal nodes in the trees and $id$ and $id'$ are the maximal internal degree of internal nodes in input tree $T$ and $T'$ respectively. Internal degree of an internal node is the number of internal nodes con-

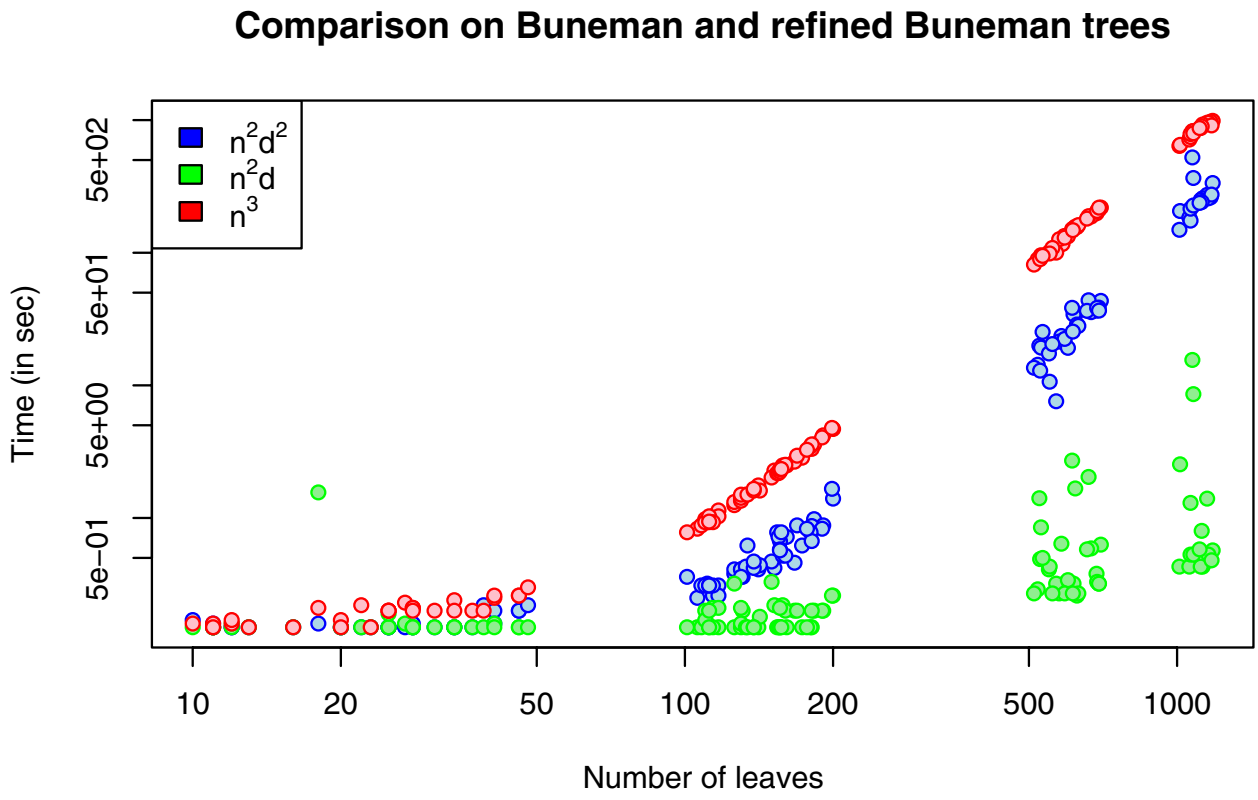


**Figure 5**
**Comparisons with earlier algorithms on Buneman and refined Buneman trees**. The running time for the new $O(n^2d)$ time algorithm compared to the existing $O(n^2d^2)$ and $O(n^3)$ time algorithms on the Buneman and the refined Buneman trees for range of Pfam based distance matrices. The plot is in log-scale on both the x- and y-axis.

nected to it, so neighbouring leaves do not add to this value. The values $|V|$, $|V'|$, $id$ and $id'$ are not independent, therefore we have investigated how the structure of the trees affect the running time of the algorithm. We show that the time used to count the butterfly quartets – topologies where one pair of the four leaves is separated from the other pair by an edge – is reduced to $O(n^2)$ when $\min\{id, id'\} = O(d_{min}^2)$, where $d_{min}$ is the minimal degree of all internal nodes in the trees. If the input trees are $d$-ary, that is all internal nodes have degree $d$, the running time is $O(\frac{n^2}{d})$, excluding the time to find intersections. These theoretical running times have been validated by running a series of tests using a Java implementation of the algorithm, available at [15]. We also done a series of tests on random trees, trees generated by the program r8s, Buneman trees, and refined Buneman trees. Running the algorithm on these trees gives an impression on how it performs on trees used in practice. On both types of trees the running time appears to be $O(n^2)$. It is however still an open problem to develop an algorithm running in time $O(n^2)$for all types of trees.

## Methods

Consider two input trees, and assume that a quartet has butterfly topology in both trees, i.e. that one pair of the four leaves is separated from the other pair by an edge in the tree in both trees. We say that the butterfly quartet is *shared*, if it has the same butterfly topology in both trees. Otherwise, we say that the butterfly quartet is *nonshared*. We let shared($T$, $T'$) denote the number of butterflies shared between tree $T$ and tree $T'$, i.e. the number of quartets that are butterflies with the same topology in tree $T$ and tree $T'$, and let nonshared ($T$, $T'$) denote the number of quartets that are butterflies in both $T$ and $T'$ but with different topology. By our definition of *shared*, the number of butterfly quartets in a single tree can be stated as the number of butterfly quartets shared between the

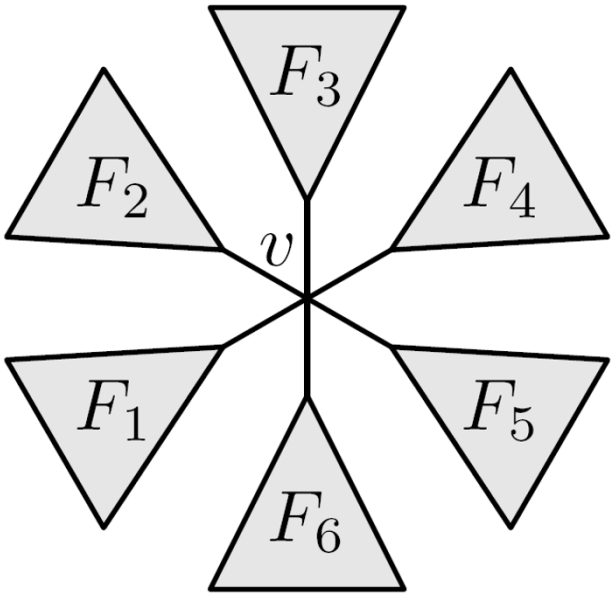


**Figure 6**
An internal node $v \in T$ with subtrees $F_1,...,F_d$, here $d_v = 6$.

tree and itself, i.e. shared($T$, $T$) or shared($T'$, $T'$) for the number of butterfly quartets in $T$ and $T'$ respectively. (This notation also emphasizes that computing the number of butterfly quartets in a single tree by our algorithm is performed as a comparison of the tree against itself.)

In [11] we argue that the quartet distance between $T$ and $T'$, qdist($T$, $T'$), can be found by focusing only on the computation of the number of shared and nonshared butterfly quartets between two trees, i.e. it is unnecessary to consider non-butterfly quartets explicitely. More specifically, we show that:

$$\begin{aligned}\text{qdist}(T,T') &= \text{shared}(T,T) + \text{shared}(T',T') \\ &\quad - 2\cdot\text{shared}(T,T') - \text{nonshared}(T,T')\end{aligned} \tag{1}$$

The proof of this formula is a follows. Let $Q$ denote the number of quartets which have butterfly topology in $T$ and non-butterfly topology in $T'$. Symmetrically, let $Q'$ denote the number of quartets which have butterfly topology in $T'$ and non-butterfly topology in T. A butterfly quartet in $T$ is either a butterfly quartet in $T'$ or a non-butterfly quartet in $T'$. The number of butterfly quartets in $T$, shared($T$, $T$), can thus be expressed as the sum shared($T$, $T'$) + nonshared($T$, $T'$) + $Q$. Similarly, the sum shared ($T'$, $T'$) = $Q'$ + shared ($T$, $T'$) + nonshared ($T$, $T'$). It is now straightforward to verify that the righthand side of (1) adds up $Q + Q'$ + nonshared($T$, $T'$) which is the number of quartets where the quartet topologies differ in $T$ and $T'$, i.e. qdist($T$,$T'$).

In the section below, we describe how to use (1) to compute the quartet distance in time $O(n + |V||V'| \min\{id, id'\})$, more precisely $O(n + |V||V'|)$ for a preprocessing step, after which we can use $O(|V||V'|)$ for calculating shared($T$, $T'$), $O(|V||V'|\{id, id'\})$ for calculating nonshared($T$,$T'$), $O(|V|)$ for calculating shared($T$,$T$) and $O(|V'|)$ for calculating shared($T'$,$T'$).

### *Terminology*

Let $T$ and $T'$ be two unrooted trees. In this paper we will explicitly refer to the leaves of a tree as *leaves* and the non-leaf nodes as *internal node*. We will assume that $T$ and $T'$ each has $n$ labelled leaves numbered 1,..., $n$ such that the leaf numbered $x$ in $T$ has the same label as the leaf numbered $x$ in $T'$. The leaf sets are denoted $L$ and $L'$ for $T$ and $T'$ respectively, note that $L = L'$. We will use $V$ and $V'$ to denote the internal nodes in $T$ and $T'$ respectively. The *degree* of an internal node $v$ is the number of subtrees connected to it, and is denoted $d_v$. The *internal degree* of an internal node $v$, $id_v$, is the number of non-leaf subtrees connected to it. We will assume that no internal node in $T$ and $T'$ has degree two, and we will denote the maximal internal degree of all internal nodes in $T$ and $T'$ by $id$ and $id'$ respectively. Let $v$ be an internal node in $T$, and let $F_1$, ..., $F_{d_v}$ be the subtrees connected to it, as shown in Fig. 6 We call these the subtrees *of v*. We say that *v claims* all butterfly quartets $ab|cd$ where $a,b \in F_i$, $c \in F_k$ and $d \in F_m$ for $i \neq k \neq m$ (see Fig. 7). With this definition, each butterfly quartet is claimed by exactly two internal nodes.

Adding the subscript $yz|w|x$ to an internal node claiming the butterfly quartet $wx|yz$, indicates that the leaves $y$ and $z$ are found in a single subtree of the internal node, while the leaves $w$ and $x$ are found in different subtrees. For example, considering the quartet $ab|cd$, $v$ and $v'$ in Fig. 7 are written as $v_{ab|c|d}$ and $v'_{ab|c|d}$.

Given a subtree $F$ of $T$, and a subtree $G$ of $T'$, we call the intersection $F \cap G$ a *shared leaf set*, i.e. the set of leaves present in both $F$ and $G$. The size of the shared leaf set, $|F \cap G|$, then denotes the number of leaves present in both $F$ and $G$. The size of a single subtree $F$ is similarly denoted $|F|$. We will use $\bar{F}$ to represent the subtree of $T$ containing all leaves not in $F$ and similarly for $\bar{G}$ and $G$ in $T'$, see Fig. 8 for an example. Note that $\bar{F}$ and $\bar{G}$ are also subtrees of $T$ and $T'$ respectively, and thus $|F \cap G|$, $|\bar{F} \cap G|$, $|F \cap \bar{G}|$ and $|\bar{F} \cap \bar{G}|$ are all sizes of shared leaf sets between a *single* subtree from $T$ and a *single* subtree from $T'$. In the pres-

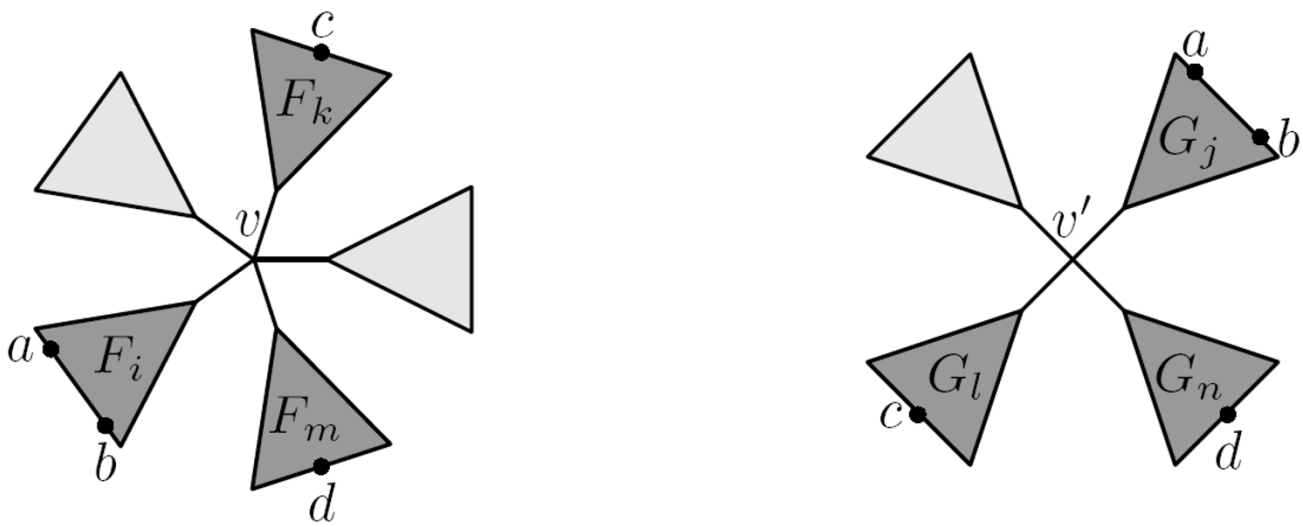


**Figure 7**
Internal nodes $v \in T$ and $v' \in T'$, each claiming the quartet $ab|cd$.

entation of the algorithms below, we will assume that we have access to $|F|$, $|G|$ and $|F \cap G|$ for all subtrees $F$ of $T$ and $G$ of $T'$ in time $O(1)$. At the end of the section we will describe how this can be achieved by an $O(n)$ time preprocessing step, which does not affect the asymptotic worst case running time of the presented algorithms.

### *Counting shared butterfly quartets*

For each pair of internal nodes $v$, $v'$ from $T$, $T'$ we want to count the number of shared butterfly quartets claimed by both internal nodes, shared($v$, $v'$). Assume that $F_1, \ldots, F_{d_v}$ are subtrees of $v$ and $G_1, \ldots, G_{d_{v'}}$ are subtrees of $v'$. We wish to count all quartets on the form $ab|cd$ where $a, b \in F_i \cap G_j$, $c \in F_k \cap G_l$ and $d \in F_m \cap G_n$, $i \neq k \neq m$, $j \neq l \neq n$ (see Fig. 7). Counting the possible combinations of $a$ and $b$ is expressed by the following double sum, which sums over all pairs of subtrees of $v$ and $v'$:

$$\sum_i \sum_j \binom{|F_i \cap G_j|}{2}$$

Given that $a$ and $b$ are in $F_i \cap G_j$, we need to find $c$ and $d$ in $\bar{F}_i \cap \bar{G}_j$. The number of possible choices of $c$ and $d$ is expressed by:

$$\binom{|\bar{F}_i \cap \bar{G}_j|}{2}$$

However when finding $c$ and $d$ in $\bar{F}_i \cap \bar{G}_j$, the condition that $c$ and $d$ must be in different subtrees is not satisfied. Therefore we subtract the number of times $c$ and $d$ are in the same subtree of $v$ and $v'$:

$$\sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} + \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} - \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2}$$

Any pair in $F_k \cap G_l$ is counted twice, once in $|F_k \cap \bar{G}_j|$ and once in $|\bar{F}_i \cap G_l|$, therefore these pairs are subtracted once using the double sum above. (2) expresses the number of ways $c$ and $d$ can be found in different subtrees, given that a and $b$ are found in $F_i \cap G_j$:

$$\binom{|\bar{F}_i \cap \bar{G}_j|}{2} - \sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} - \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} + \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2} \qquad (2)$$

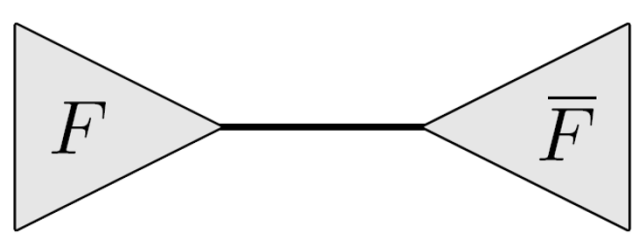


**Figure 8**
A rooted subtree $F$, and its complement rooted subtree $\bar{F}$.

We can now compute the number of shared butterfly quartets between two internal nodes, ie. the number of butterfly quartets claimed by both internal nodes with the same topology:

$$\begin{aligned}\text{shared}(v,v') = \sum_i \sum_j \binom{|F_i \cap G_j|}{2} \Bigg( \binom{|\bar{F}_i \cap \bar{G}_j|}{2} - \sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} \\ - \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} + \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2} \Bigg)\end{aligned} \tag{3}$$

If the trees, $T$ and $T'$, have a shared quartet $ab|cd$, then there are two internal nodes in each tree that claims this quartet: $v_{ab|c|d}$ and $v_{cd|a|b}$ in $T$ and $v'_{ab|c|d}$ and $v'_{cd|a|b}$ in $T'$. Since both shared($v_{ab|c|d}$, $v'_{ab|c|d}$) and shared($v_{cd|a|b}$, $v'_{cd|a|b}$) will count the quartet, the total number of shared quartets between the two trees is:

$$\text{shared}(T,T') = \frac{1}{2} \sum_{v \in T} \sum_{v' \in T'} \text{shared}(v,v')$$

It is straightforward to observe that calculating shared($v$, $v'$) using a direct computation of (3) takes time $O(d_v^2 d_{v'}^2)$. It is however not necessary for shared($v$, $v'$) to sum over all subtrees of $v$ and $v'$. Since each term in the sums involves taking a 2-subset from a shared leaf set, we need only to consider subtrees that are not leaves. This reduces the running time to $O(id_v^2 id_{v'}^2)$. This running time can be improved even more, we start by expressing (2) in a different way:

$$\begin{aligned}&\underbrace{\binom{|\bar{F}_i \cap \bar{G}_j|}{2}}_{I} - \underbrace{\sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2}}_{II} - \underbrace{\sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2}}_{III} + \underbrace{\sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2}}_{IV} = \\ &\underbrace{\binom{|\bar{F}_i \cap \bar{G}_j|}{2}}_{I} - \underbrace{\sum_k \binom{|F_k \cap \bar{G}_j|}{2} + \binom{|F_i \cap \bar{G}_j|}{2}}_{II} - \underbrace{\sum_l \binom{|\bar{F}_i \cap G_l|}{2} + \binom{|\bar{F}_i \cap G_j|}{2}}_{III} \\ &+ \underbrace{\sum_k \sum_l \binom{|F_k \cap G_l|}{2} - \sum_k \binom{|F_k \cap G_j|}{2} - \sum_l \binom{|F_i \cap G_l|}{2} + \binom{|F_i \cap G_j|}{2}}_{IV}\end{aligned} \tag{4}$$

Let

$$\begin{aligned}S_j &= \sum_k \binom{|F_k \cap G_j|}{2} \\ S'_i &= \sum_l \binom{|F_i \cap G_l|}{2} \\ \bar{S}_j &= \sum_k \binom{|F_k \cap \bar{G}_j|}{2} \\ \bar{S}'_i &= \sum_l \binom{|\bar{F}_i \cap G_l|}{2} \\ S &= \sum_k \sum_l \binom{|F_k \cap G_l|}{2}\end{aligned} \tag{5}$$

We can ignore leaf subtrees, so we need to compute $id_{v'}$ different $\bar{S}_j$'s and $S_j$'s which can each be computed in $O(id_v)$ time. Symmetrically each of the $id_v$ $\bar{S}'_i$'s and $S'_i$'s takes time $O(id'_v)$ to compute, and the total time of computing $S$ is $O(id_v id_{v'})$. The total time of computing all sums mentioned is thus $O(id_v id_{v'})$ and this is the key to reducing the time usage of shared($v$,$v'$). Using the sums we can express (4) as:

$$\underbrace{\binom{|\bar{F}_i \cap \bar{G}_j|}{2}}_{I} - \underbrace{\bar{S}_j + \binom{|F_i \cap \bar{G}_j|}{2}}_{II} - \underbrace{\bar{S}'_i + \binom{|\bar{F}_i \cap G_j|}{2}}_{III} + \underbrace{S - S_j - S'_i + \binom{|F_i \cap G_j|}{2}}_{IV}$$

Provided that the sums $\bar{S}_j$, $\bar{S}'_i$, $S_j$, $S'_i$ and $S$ have been calculated, (4) can be calculated in time $O(1)$. Since calculation of the sums is independent on the calculation of shared($v$, $v'$), these calculations can be done serially as shown in the algorithm below, thereby reducing the time usage of shared($T$, $T'$) to:

$$\sum_{v \in T} \sum_{v' \in T'} id_v id_{v'} = \sum_{v \in T} id_v \sum_{v' \in T'} id_{v'} = 2(|V|-1) \cdot 2(|V'|-1) = O(|V||V'|)$$

ALGORITHM – CALCULATING THE NUMBER OF SHARED BUTTERFLY QUARTETS BETWEEN $T$ AND $T'$

**Requires:** $T$, $T'$ two input trees with the same leaf set.

**Ensures:** $Res$ = shared($T$, $T'$)

$Res \leftarrow 0$

**for** $v$ internal node in $T$ **do**

  **for** $v'$ internal node in $T'$ **do**

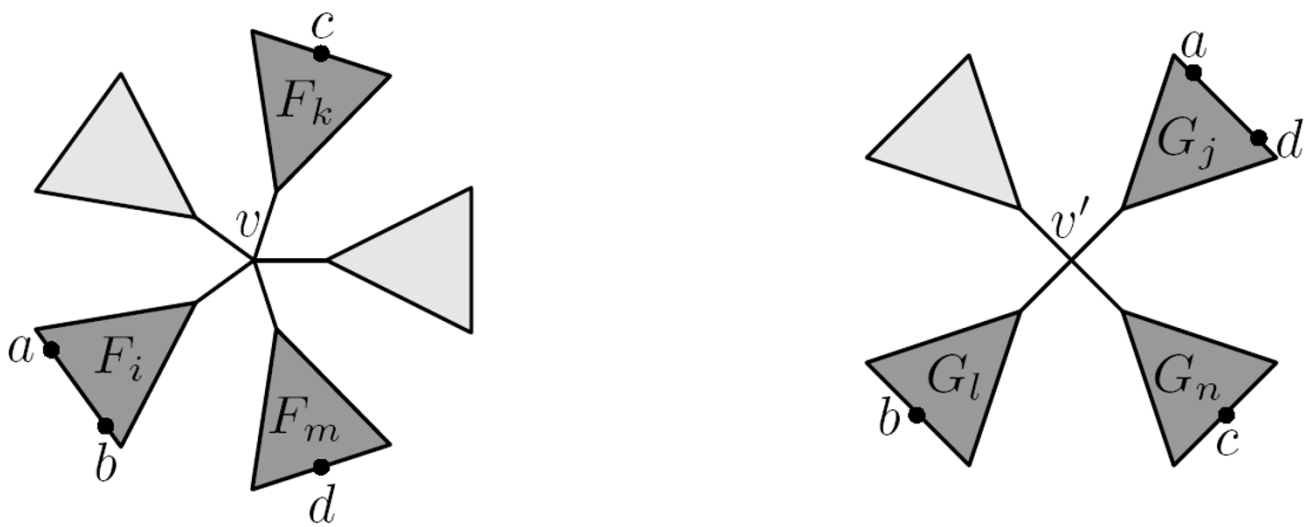


**Figure 9**
Internal nodes $v \in T$ claiming the quartet $ab|cd$ and $v' \in T'$ claiming the quartet $ad|bc$.

Calculate sums $\bar{S}_j$, $\bar{S}'_i$, $S_j$, $S'_i$ and $S$

$Res \leftarrow Res + \text{shared}(v, v')$

**end for**

**end for**

$$Res \leftarrow \frac{Res}{2}$$

***Counting nonshared butterfly quartets***

For each pair of internal nodes $v$, $v'$ we want to count the number of nonshared butterfly quartets claimed by both internal nodes, nonshared($v$, $v'$). Such quartets have the property that a pair of leaves found in the same subtree of $v$ will be found in different subtrees of $v'$ and vice versa, i.e. a nonshared quartet with leaves $a$, $b$, $c$ and $d$, has a $\in F_i \cap G_j$, $b \in F_i \cap G_l$, $c \in F_k \cap G_n$ and $d \in F_m \cap G_j$ (see Fig. 9). The following expression counts all nonshared quartets related to a pair of nodes $v$ and $v'$, obeying that if two leaves of the quartet are in one subtree of $v$ they are in different subtrees of $v'$ and vice versa:

$$\sum_i \sum_j |F_i \cap G_j||F_i \cap \bar{G}_j||\bar{F}_i \cap \bar{G}_j||\bar{F}_i \cap G_j| \quad (6)$$

Even though (6) satisfies the property of nonshared quartets, it possibly counts more than the number of nonshared quartets claimed by an internal node in each tree. The problem is that given two internal nodes, they do not nescessarily *claim* the quartets counted by (6). If we denote the leaves of an nonshared quartet $a$, $b$, $c$ and $d$, the first, second, third and fourth factors in (6) counts the number of choices of $a$, $b$, $c$ and $d$ respectively. The first and second factor choose $a$ and $b$ from $F_i$, while the third and fourth choose $c$ and $d$ from $\bar{F}_i$. In the cases where $c$ and $d$ are chosen from the same subtree $F_k$, $k \neq i$ of $v$, $v$ does not claim the quartet. We must subtract these quartets, which can be counted as:

$$\sum_i \sum_j \sum_{k \neq i} |F_i \cap G_j||F_i \cap \bar{G}_j||F_k \cap \bar{G}_j||F_k \cap G_j| \quad (7)$$

Similarly there are cases where $b$ and $c$ are chosen from the same subtree $G_l$, $l \neq j$ of $v'$, which we must also subtract. These can be counted as:

$$\sum_i \sum_j \sum_{l \neq j} |F_i \cap G_j||F_i \cap G_l||\bar{F}_i \cap G_l||\bar{F}_i \cap G_j| \quad (8)$$

The cases where both $c$ and $d$ are chosen from the same subtree $F_k$, $k \neq i$ of $v$ and $b$ and $c$ are chosen from the same subtree $G_l$, $l \neq j$ of $v'$ are included in both the expressions above and therefore they must be added again. The following expression counts the number of these cases:

$$\sum_i \sum_j \sum_{k \neq i} \sum_{l \neq j} |F_i \cap G_j||F_i \cap G_l||F_k \cap G_l||F_k \cap G_j| \quad (9)$$

Combining equations (6), (7), (8) and (9), gives a way of calculating the number of nonshared quartets between two internal nodes $v$ and $v'$:

$$\begin{aligned}\text{nonshared}(v,v') = \sum_i \sum_j \Big( & |F_i \cap G_j||F_i \cap \bar{G}_j||\bar{F}_i \cap \bar{G}_j||\bar{F}_i \cap G_j| \\ & -\sum_{k\neq i} |F_i \cap G_j||F_i \cap \bar{G}_j||F_k \cap \bar{G}_j||F_k \cap G_j| \\ & -\sum_{l\neq j} |F_i \cap G_j||F_i \cap G_l||\bar{F}_i \cap G_l||\bar{F}_i \cap G_j| \\ & +\sum_{k\neq i}\sum_{l\neq j} |F_i \cap G_j||F_i \cap G_l||F_k \cap G_l||F_k \cap G_j| \Big)\end{aligned} \tag{10}$$

Assuming that the trees have a nonshared quartet with topology form $ab|cd$ in $T$, and $ad|bc$ in $T'$, there are two internal nodes in each tree claiming the quartet: $v_{ab|c|d}$ and $v_{cd|a|b}$ in $T$ and $v'_{ad|b|c}$ and $v'_{bc|a|d}$ in $T'$. All of the four combinations of these will identify the quartet as nonshared. Therefore the total number of nonshared quartets between the two trees is:

$$\text{nonshared}(T,T') = \frac{1}{4}\sum_{v\in T}\sum_{v'\in T'} \text{nonshared}(v,v')$$

Direct computation of nonshared($v$, $v'$) using (10) takes time $O(d_v^2 d_{v'}^2)$. Each of the subtrees has to be intersected with two disjoint sets in each of the sums. This means that if a subtree is only a leaf, at least one of these intersections will be zero, and the term will be zero. Therefore we can ignore subtrees that consist of a single leaf, just like when computing shared($T$, $T'$), and reduce the time usage to $O(id_v^2 id_{v'}^2)$. The time usage of the calculation of nonshared($v$, $v'$) can be further improved by rewriting (9):

$$\begin{aligned}&\sum_i\sum_j\sum_{k\neq i}\sum_{l\neq j}|F_i \cap G_j||F_i \cap G_l||F_k \cap G_l||F_k \cap G_j| = \\ &\sum_i\sum_j|F_i \cap G_j|\sum_{k\neq i}|F_k \cap G_j|\sum_{l\neq j}|F_i \cap G_l||F_k \cap G_l| = \\ &\sum_i\sum_j|F_i \cap G_j|\sum_{k\neq i}|F_k \cap G_j|\left(\sum_l |F_i \cap G_l||F_k \cap G_l| - |F_i \cap G_j||F_k \cap G_j|\right)\end{aligned}$$

Inspired by the precomputing of sums used in shared($v$, $v'$), (5), we calculate for each $i$, $k$, $k \neq i$ the sum:

$$S_{i,k} = \sum_l |F_i \cap G_l||F_k \cap G_l| \tag{11}$$

There are $O(id_v^2)$ of these sums and each takes time $O(id_{v'})$ to calculate, so the time complexity for calculating all sums is $O(id_v^2 id_{v'})$. In the case that $id_{v'} \leq id_{v'}$ we can switch $v$ and $v'$ and thus get time usage $O(id_v id_{v'} \min\{id_v, id_{v'}\})$. Assuming that the sums have been calculated, (9) can now be calculated in time $O(id_v id_{v'} \min\{id_v, id_{v'}\})$ by the expression:

$$\sum_i\sum_j|F_i \cap G_j|\sum_{k\neq i}|F_k \cap G_j|\left(S_{i,k} - |F_i \cap G_j||F_k \cap G_j|\right) \tag{12}$$

By substituting (9) with (12) in (10), we can calculate nonshared($v$, $v'$, in time $O(id_v id_{v'} \min\{id_v, id_{v'}\})$. Since calculation of the sums is independent of the calculation of nonshared($v$, $v'$), these calculations can be done serially as shown in the algorithm below.

ALGORITHM – CALCULATING THE NUMBER OF NONSHARED BUTTERFLY QUARTETS BETWEEN $T$ AND $T'$

**Requires:** $T$, $T'$ two input trees with the same leaf set.

**Ensures:** $Res$ = nonshared($T$, $T'$)

$Res \leftarrow 0$

**for** $v$ internal node in $T$ **do**

  **for** $v'$ internal node in $T'$ **do**

    Calculate sums $S_{i,k}$

    $Res \leftarrow Res$ + nonshared($v$, $v'$)

  **end for**

**end for**

$$Res \leftarrow \frac{Res}{4}$$

The time complexity of the algorithm is:

$$\sum_{v\in T}\sum_{v'\in T'} id_v id_{v'} \min\{id_v, id_{v'}\} \leq \min\{id, id'\}\sum_{v\in T} id_v \sum_{v'\in T'} id_{v'} = O(|V||V'|\min\{id, id'\})$$

#### *Counting butterfly quartets in a single tree*

Reusing the idea of precomputing certain sums enables us to calculate the number of butterfly quartets in a single tree $T$ in time $O(|V|)$. Since the number of butterfly quartets in a single tree is the number of butterfly quartets shared between the tree and itself, we will use shared($T$, $T$) to denote the number of butterfly quartet in $T$. This notation also emphasizes that computing the number is essentially a comparison of the tree against itself. Given a node $v$ in $T$ we can express the number of quartets it claims in the following way:

$$\sum_i \binom{F_i}{2}\left(\binom{\bar{F}_i}{2} - \sum_{j\neq i}\binom{F_j}{2}\right) \tag{13}$$

where the $F_i$'s are the subtrees of $v$. Now let

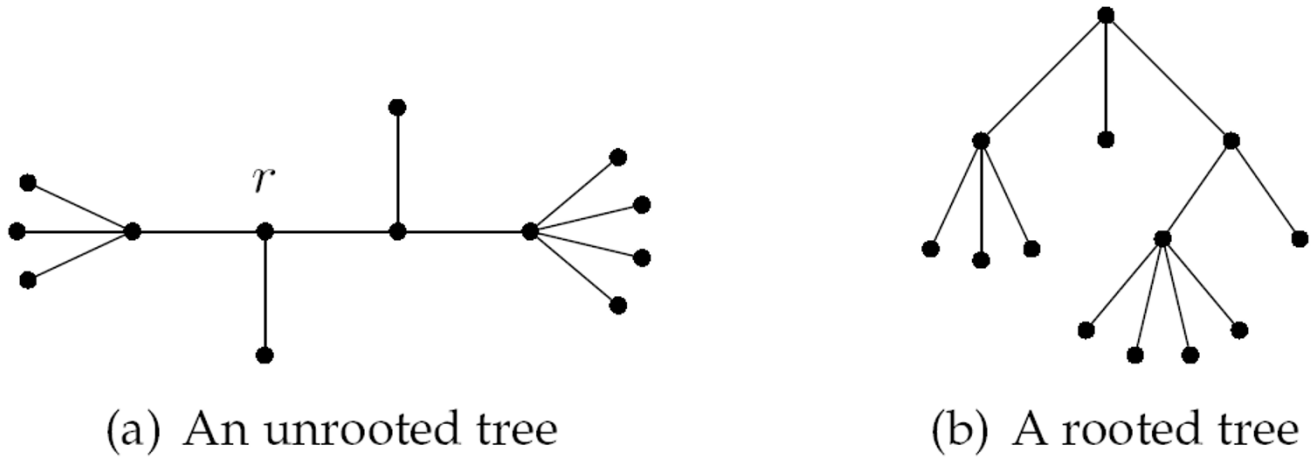


**Figure 10**
An arbitrary node, *r*, is chosen as the root in the tree, leading to a rooted tree.

$$S = \sum_j \binom{F_j}{2},$$

we can now express (13) as

$$\sum_i \binom{F_i}{2}\left(\binom{\bar{F}_i}{2} - S + \binom{F_i}{2}\right). \qquad (14)$$

$S$ can be calculated in time $O(id_v)$and using the precomputed $S$, (14) can be also calculated in time $O(id_v)$. Summing the results of (14) for all nodes in $T$ gives the number of quartets in the tree, shared($T$, $T$). The total time usage is

$$\sum_{v \in T} id_v = O(|V|).$$

***Calculating the shared leaf set sizes***

The algorithms presented above all rely on $O(1)$ time access to the size of the shared leaf set $|F \cap G|$ for any pair of subtrees $F$ of $T$ and $G$ of $T'$, where $F$ and $G$ each has size at least two, i.e. contains more than a single leaf. We will refer to $|F \cap G|$ as the intersection size of subtree $F$ and $G$. In [9] and $O(n^2)$ time and space algorithm is presented for computing the intersection sizes of all pairs of subtrees $F$ and $G$ of two binary input trees. A straightforward generalization of this algorithm to two input trees $T$ and $T'$ of arbitrary degrees results in an $O(n^2dd')$ time and $O(n^2)$ space algorithm, which gives a worst case running time of $O(n^4)$.

In this section we will present an improved algorithm for computing the intersection sizes of all pairs of subtrees of $T$ and $T'$ which runs in time $O(n + |V||V'|)$ and space $O(|V||V'|)$. We will assume that the size of each subtree $F$ of $T$ and $G$ of $T'$, i.e. $|F|$ and $|G|$, is available in time $O(1)$. This can be achieved as presented in the next section. Our algorithm for computing all intersection sizes is as follows. Choose an arbitrary node $r$ in $T$ and an arbitrary node $r'$ in $T'$. Rooting the trees in $r$ and $r'$ respectively gives rise to two rooted trees $T_r$ and $T'_r$, Fig. 10 shows an example of rooting a tree. Calculating the shared leaf set sizes of $T_r$ and $T'_r$ and all subtrees in both trees can be done using:

$$|T_r \cap T'_r| = \sum_i \sum_j |F_i \cap G_j| = \sum_i |F_i \cap G| = \sum_j |F \cap G_j|, \qquad (15)$$

where $F_i$ are all subtrees of $T_r$ and $G_j$ are all subtrees of $T'_r$. This can be calculated using dynamic programming in time $O(n^2)$:

$$\sum_{v \in V} \sum_{v' \in V'} d_v d_{v'} + \sum_{v \in V} \sum_{l' \in L'} d_v + \sum_{l \in L} \sum_{v' \in V'} d_{v'} + \sum_{l \in L} \sum_{l' \in L'} 1 = O(n^2)$$

Except $|T_r \cap T'_r|$, the shared leafset sizes calculated by (15) are the shared leaf set sizes of all rooted subtrees of $T$ and $T'$ that do not contain the nodes $r$ and $r'$. Assuming that the subtree $F$ of $T$ does *not* contain $r$, then the subtree $\bar{F}$ *does* contain $r$ and similarly for $r'$ and subtrees $G$ and $\bar{G}$ of $T'$. The shared leaf set sizes of these trees that *do* contain $r$ and $r'$ can be calculated from the intersection sizes that we have available using (16):

$$\begin{aligned} |\bar{F} \cap G| &= |G| - |F \cap G| \\ |F \cap \bar{G}| &= |F| - |F \cap G| \\ |\bar{F} \cap \bar{G}| &= n - (|G| + |F| - |F \cap G|) \end{aligned} \quad (16)$$

In other words all shared leaf set sizes can be calculated in time $O(n^2)$. First the shared leaf set sizes for subtrees that to not contain an arbitrary node $r$ and $r'$ from each tree are calculated in time $O(n^2)$. Then the shared leaf set sizes of subtrees that do contain $r$ or $r'$ (or both) are calculated constant time for each shared leaf set. Since there are $O(n^2)$ shared leaf sets the total time usage is $O(n^2)$.

The reduction to time $O(n + |V||V'|)$ and space $O(|V||V'|)$ is done by handling the cases where $F$, $F_i$, $G$ or $G_j$ is a leaf in a special way. For each pair of nodes $v$, $v'$ we let *Leaf* $[v, v']$ be the number of leaves directly connected to $v$ that have the same label as a leaf directly connected to $v'$. *Leaf* $[v, v']$ is constructable in time $O(n + |V||V'|)$ in the following way: First, set *Leaf* $[v, v'] = 0$ for all pairs of nodes $v$, $v'$. Given a leaf number, $x$, there is a unique node, *node*$(x)$, in $T$ and a unique node, *node*$'(x)$, in $T'$. For each leaf number, $x$, we increment *Leaf* $[node(x), node'(x)]$. There are $n$ such numbers, and by assumption, the leaves can be found in constant time given a number. Thus *Leaf* $[v, v']$ is constructable in time $O(n + |V||V'|)$. We choose $r$ and $r'$ in $T$ and $T'$ and create two rooted trees $T_r$ and $T'_r$. The *non-leaf* children of $r$ in $T_r$ are $F_1,\ldots, F_x$ and the *non-leaf* children of $r'$ in $T'_r$ are $G_1,\ldots, G_y$. The intersection size of the two trees can be defined recursively as:

$$|T_r \cap T'_r| = Leaf[r, r'] + \sum_{i=1}^{x} |F_i \cap T'_r| + \sum_{j=1}^{y} |T_r \cap G_j| - \sum_{i=1}^{x}\sum_{j=1}^{y} |F_i \cap G_j| \quad (17)$$

The first term counts all leaves directly connected to both $r$ and $r'$. The second term counts all leaves connected directly to $r'$, that are also in $T_r$, but not directly connected to $r$. The third term counts all leaves connected directly to $r$, that are also in $T'_r$, but not directly connected to $r'$. Summing these three terms counts all leaves present in both subtrees, but leaves not connected directly to the roots are counted twice, and are subtracted by the last term.

Since (17) is only summing over the *non-leaf* children of a given internal node, calculating the shared leaf set sizes of all these pairs of subtrees can be done using dynamic programming in time:

$$(n + |V||V'|) + \sum_{v \in V} id_v + \sum_{v' \in V'} id_{v'} + \sum_{v \in V}\sum_{v' \in V'} id_v id_{v'} = O(n + |V||V'|)$$

By the same arguments as above, the rest of the shared leaf set sizes can be computed in time $O(|V||V'|)$ and space $O(|V||V'|)$. Therefore the total running time of the algorithm is $O(n + |V||V'|)$ and space usage is $O(|V||V'|)$.

#### *Calculating the sizes of all subtree leaf sets*

All of the above algorithms make use of the sizes of the leaf sets of the rooted subtrees of the input trees, either directly or indirectly. Rooting $T$ in an arbitrary node $r$ gives rise to the rooted tree $T_r$. Every subtree $F_x$ of $T_r$ is a rooted subtree of $T$, and $\bar{F}_x$ is also a rooted subtree of $T$. Note that the set of subtrees $F_x \cup \bar{F}_x$, since one tree is the complement of the other, contains all subtrees of $T$ and that $|\bar{F}_x| = n - |F_x|$. By using dynamic programming the sizes of all subtrees, $F_x$, can be computed by a single traversal of $T_r$. For each $F_x$ the size of $\bar{F}_x$ can be computed in constant time, since $n$ is known. This means that all leaf set sizes of a tree of arbitrary degree can be calculated in time $O(n)$.

## Authors' contributions

All authors participated in developing the algorithm. CC and MR implemented the algorithm and conducted the experiments. All authors participated in drafting of the manuscript.

## References


1. Felsenstein J: *Inferring Phylogenies* Sinauer Associates Inc; 2004.
2. Robinson DP, Foulds LR: **Comparison of weighted labelled trees.** In *Combinatorial mathematics, VI (Proc 6th Austral Conf)* Lecture Notes in Mathematics, Springer; 1979:119-126.
3. Waterman MS, Smith TF: **On the similarity of dendrograms.** *Journal of Theoretical Biology* 1978, **73:**789-800.
4. Allen BL, Steel M: **Subtree transfer operations and their induced metrics on evolutionary trees.** *Annals of Combinatorics* 2001, **5:**1-13.
5. Robinson DP, Foulds LR: **Comparison of phylogenetic trees.** *Mathematical Biosciences* 1981, **53:**131-147.
6. Estabrook G, McMorris F, Meacham C: **Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units.** *Syst Zool* 1985, **34:**193-200.
7. Steel M, Penny D: **Distribution of tree comparison metrics–some new results.** *Syst Biol* 1993, **42(2):**126-141.
8. Doucette CR: **An Efficient Algorithm to Compute Quartet Dissimilarity Measures.** 1985. [Unpublished, Bachelor of Science (Honours) Dissertation. Memorial University of Newfoundland]
9. Bryant D, Tsang J, Kearney PE, Li M: **Computing the quartet distance between evolutionary trees.** *Proceedings of the 11th Annual Symposium on Discrete Algorithms (SODA)* 2000:285-286.
10. Brodal GS, Fagerberg R, Pedersen CNS: **Computing the Quartet Distance Between Evolutionary Trees in Time *O*(*n* log *n*).** *Algorithmica* 2003, **38:**377-395.
11. Christiansen C, Mailund T, Pedersen CNS, Randers M: **Computing the Quartet Distance Between Trees of Arbitrary Degree.** In *Proceedings of Workshop on Algorithms in Bioinformatics (WABI) Volume 3692*. LNBI, Springer-Verlag; 2005:77-88.
12. **r8s** [http://ginger.ucdavis.edu/r8s/]
13. **Pfam** [http://www.sanger.ac.uk/Software/Pfam/]
14. Besenbacher S, Mailund T, Westh-Nielsen L, Pedersen CNS: **RBT – A tool for building refined Buneman trees.** *Bioinformatics* 2005, **21:**1711-1712.
15. **QuartetDist** [http://www.daimi.au.dk/~chrisc/qdist/]