A sub-cubic time algorithm for computing the quartet distance between two general trees

Jesper Nielsen¹²*, Anders K Kristensen², Thomas Mailund¹ and Christian NS Pedersen¹²

Abstract

Background: When inferring phylogenetic trees different algorithms may give different trees. To study such effects a measure for the distance between two trees is useful. Quartet distance is one such measure, and is the number of quartet topologies that differ between two trees.

Results: We have derived a new algorithm for computing the quartet distance between a pair of general trees, i.e. trees where inner nodes can have any degree ≥ 3. The time and space complexity of our algorithm is sub-cubic in the number of leaves and does not depend on the degree of the inner nodes. This makes it the fastest algorithm so far for computing the quartet distance between general trees independent of the degree of the inner nodes.

Conclusions: We have implemented our algorithm and two of the best competitors. Our new algorithm is significantly faster than the competition and seems to run in close to quadratic time in practice.

Background

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 inference methods to infer such trees from biological data, or using different biological data from the same set of species, often yield slightly different trees. 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 this 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 [1], the nearest-neighbour interchange [2], the subtree transfer distance [3], the Robinson and Foulds distance [4], and the quartet distance [5]. Each distance measure has different properties and reflects different properties of the tree relationship.

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 Figure 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.

Most previous work has focused on comparing binary trees and therefore avoided star quartets. Steel and Penny in [6] developed an algorithm for computing the quartet distance in time O(n³). Bryant et al. in [7] improved this result with an algorithm that computes the quartet distance in time O(n²). Brodal et al., in [8], presented the currently best known algorithm that algorithm the computes the quartet distance in time O(n log n).

Recently, we have developed algorithms for computing the quartet distance between two trees of arbitrary degrees, i.e. trees that can contain star quartets. In [9] we developed two algorithms: the first algorithm runs in time O(n²d²) and space O(n³)–and is thus independent of the degree of the inner nodes—the second in time O(n²d²) and space O(n³), where d is the maximal degree of inner nodes in the trees—and thus depends on the degree of the nodes. The O(n²d²) was later improved to O(n²d) [10], and by taking an approach similar to the Brodal et al. [8] O(n log n) we developed a sub-quadratic algorithm in terms of n but at a significant cost in terms of d: O(d²n log n) [11].

* Correspondence: jn@birc.au.dk

1Bioinformatics Research Centre (BiRC), Aarhus University, C. F. Møllers Alle 8, DK-8000 Aarhus C, Denmark

© 2011 Nielsen et al; licensee BioMed Central Ltd. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/2.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
In this paper we develop an $O(n^{2+\omega})$ algorithm, where \( \alpha = \frac{\omega - 1}{2} \) and \( O(n^\alpha) \) is the time it takes to multiply two \( n \times n \) matrices. Using the Coppersmith-Winograd [12] algorithm, where \( \omega = 2.376 \), this yields a running time of \( O(n^{2.688}) \). The running time is thus independent of the degrees of the inner nodes of the input trees, and this is the first sub-cubic time algorithm with this property. Furthermore we have implemented the algorithm, along with two of the previous methods, and show experimentally that our new algorithm performs well in practice.

Methods: A sub-cubic time and space algorithm

The quartet distance between two trees is the number of quartets where the quartet topology differs between the two trees, i.e. the number of quartets where one tree has the star topology and the other a butterfly topology, plus the number of quartets where the trees have a different butterfly topology. As observed in [9], the former—where one tree has the star topology and the other a butterfly topology—can be expressed in terms of the total number of butterflies in the two trees, the number of shared butterflies and the number of different butterflies: For trees \( T \) and \( T' \), the number of different topologies due to one being a star and the other a quartet, \( \text{diff}_S(T, T') \), is given by

\[
\text{diff}_S(T, T') = B + B' - 2(\text{shared}_B(T, T') + \text{diff}_B(T, T')).
\]  

where \( B \) is the number of butterflies in \( T \), \( B' \) the number of butterflies in \( T' \), \( \text{shared}_B(T, T') \) the number of quartets with the same butterfly topology in \( T \) and \( T' \) and \( \text{diff}_B(T') \) the number of quartets with different butterfly topologies in \( T \) and \( T' \). Thus the quartet distance between \( T \) and \( T' \) is given by the expression

\[
\text{qdist}(T, T') = B + B' - \text{2shared}_B(T, T') - \text{diff}_B(T, T').
\]  

Since, \( B = \text{shared}_B(T, T') \) and \( B' = \text{shared}_B(T', T) \), an algorithm for computing \( \text{shared}_B(T, T') \) and \( \text{diff}_B(T, T') \) gives an algorithm for computing the quartet distance between \( T \) and \( T' \).

Our approach to counting the shared and different quartets is based on directed quartets and claims [8,9]. An (undirected) butterfly quartet topology, \( ab|cd \) induces two directed quartet topologies \( ab \to cd \) and \( ab \leftarrow cd \), by the orientation of the middle edge of the topology, as shown in Figure 2. There are twice as many directed butterflies as undirected. If \( e = (s, t) \) is a directed edge from \( s \) to \( t \), we call \( s \) the source of \( e \) and \( t \) the target. To each directed quartet, \( ab \to cd \), we can uniquely associate the directed edge, \( e \) so that \( a \) and \( b \) are leaves in the subtree rooted at \( s_e \) and \( c \) and \( d \) are leaves in different subtrees rooted at \( t_e \), see Figure 3. We call such a tree substructure, consisting of a directed edge \( e \) with a subtree, \( A \) behind \( e \) and two distinct subtrees, \( C \) and \( D \), in front of a \( e \) claim, written \( e \xrightarrow{A} (C, D) \). We say that the edge \( e \) claims the directed quartet \( ab \to cd \), and we also say that an edge \( e \) claims an undirected quartet \( ab|cd \) if it claims one of its directed quartets. Each (undirected) butterfly quartet defines exactly two directed butterfly quartets, and each directed quartet is claimed by exactly one directed edge; considering each claim and implicitly each directed butterfly claimed by the claim, we can examine each directed butterfly in a tree, or each undirected butterfly twice.

The crux of the algorithm is to consider each pair of claims, one from each tree, and for each such pair count the number of shared and different directed butterflies.
claimed in the two trees. This way each shared butterfly is counted twice, and each different butterfly is counted four times, as shown in Figure 4. Dividing the counts by two and four, respectively, gives us shared 

\[ B(T, T') \] and different 

\[ B(T, T') \].

Preprocessing

Before counting shared and different butterflies, we calculate a number of values in two preprocessing steps. First, we calculate a matrix that for each pairs of subtrees \( F \in T \) and \( G \in T' \) stores the number of leaves in both trees, \( |F \cap G| \). This can be achieved in time and space \( O(n^2) \) [7].

Next, for each pair of inner nodes, \( v \in T, v' \in T' \) with sub-trees \( F, i = 1, \ldots, d_v \) and \( G, j = 1, \ldots, d_{v'} \), respectively, we calculate a matrix, \( I \), such that \( I[i, j] = |F_i \cap G_j| \), and we calculate vectors of its row and column sums, and the total sum of its entries:

\[
R[i] = \sum_{j=1}^{d_v} I[i, j] \\
C[j] = \sum_{i=1}^{d_v} I[i, j] \\
M = \sum_{i=1}^{d_v} \sum_{j=1}^{d_{v'}} I[i, j]
\]

Inspired by the sums (S.3) - (S.6) in Additional file 1 we calculate a matrix \( I' \), vectors of its row and column sums, the total sum of its entries, and some further values

\[
I'[i, j] = I[i, j](M - R[i] - C[j] + I[i, j]) \\
R'[i] = \sum_{j=1}^{d_{v'}} I'[i, j] \\
C'[j] = \sum_{i=1}^{d_v} I'[i, j] \\
M' = \sum_{i=1}^{d_v} \sum_{j=1}^{d_{v'}} I'[i, j] \\
R''[i] = \sum_{j=1}^{d_{v'}} I[i, j](C[j] - I[i, j])
\]

Figure 3 A claim. A claim \( A \xrightarrow{e} (C, D) \) claims all ordered butterflies \( ab \rightarrow cd \) where \( a, b \in A \) and \( c, d \in D \) where \( C \) and \( D \) are two different subtrees in front of \( e \).

Figure 4 Counting directed claims. A shared butterfly induces two butterflies in each tree, which will give four pairs of claims, however the butterflies will only be identical in two of these pairs, thus a shared butterfly will be counted twice. A different butterfly also induces four pairs of claims, but since we are counting different butterflies all four will be counted. The way we count shared butterflies prevents the two different butterflies induced by the shared (undirected) butterfly from being counted.
and F for some claims k for all distinct l. Eq. (11)
\[ C''[j] = \sum_{i=1}^{d_r} I[i, j][R[i] - I[i, j]] \] (11)

\[ R''[i] = \sum_{j=1}^{d_r} I[i, j]^2 \] (12)

\[ C'''[j] = \sum_{i=1}^{d_r} I[i, j]^2 \] (13)

Calculating the values in Eq. (3) - (13) can be done in
O(d_r d_s) for each pair of inner nodes (v, v') ∈ T × T', giving
a total time of
\[ O \left( \sum_{v \in T} \sum_{v' \in T'} d_v d_{v'} \right) = O \left( \left( \sum_{v \in T} d_v \right) \left( \sum_{v' \in T'} d_{v'} \right) \right) = O(n^2). \]

Finally, we need to calculate the following values:
\[ l''''[i, j] = \sum_{k=1}^{d_v} \sum_{l=1}^{d_{v'}} \sum_{i=1}^{R''[i]} I[i, l] I[k, j] I[l, k] \] (14)

which takes time O(d_v^2 d_{v'}) for each pair of inner
nodes, giving a total time of O(n^2), if done naively. However, as we show in section 1 of Additional file 1 the values in Eq. (14) can be calculated faster if we pre-
compute either l''''_1 = l'''' and l''''_2 = l''''_1, or l''''_1 = l'''' and l''''_2 = l''''_1, depending on which pair of matrices is fastest to
compute, where l is the d_v × d_{v'} matrix defined above. We thus calculate either Eq. (15) and (16), or Eq.
(17) and (18), depending on which pair is fastest to cal-
culate.
\[ l''''_1[i, k] = \sum_{j=1}^{d_v} I[i, j] I[k, j] \] (15)

\[ l''''_1[i, j] = \sum_{k=1}^{d_{v'}} I[k, j] l''''_1[i, k] \] (16)

\[ l''''_2[i, j] = \sum_{i=1}^{l''''_1} I[i, j] I[i, l] \] (17)

\[ l''''_2[i, j] = \sum_{l=1}^{l''''_1} I[i, l] l''''_2[i, l] \] (18)

Calculating the values in Eq. (15) and (16) takes time
O(max(d_v d_{v'})^o) if padding the matrices to become square and with o = 2.376 if using the Coppersmith-
Winograd algorithm [12] for matrix multiplication, or
time O(d_v^2 d_{v'}) if using naive matrix multiplication. Similarly, calculating the values in Eq. (17) and (18) takes time O(max(d_v d_{v'})^o) or O(d_v d_{v'}^2). Computing either l''''_1
and l''''_2, or l''''_2 and l''''^o, thus takes time
O(min(max(d_v, d_{v'})^o, d_v^2 d_{v'}, d_v d_{v'}^2)).

Counting shared butterfly topologies
For each pair of inner edges, e ∈ T and e' ∈ T', see Fig-
ure 5, we count the directed butterflies claimed by both e
and e'. These are all on the form ab → cd, where a, b, c,
e' ∈ F_1 ∩ G_p, c ∈ F_k ∩ G_l and d ∈ F_m ∩ G_n for some claims,
F_1 → (F_k, F_m) and G_p → (G_l, G_n) of e and e'. The total
number of different butterflies common for both e and
e' is therefore given by the expression
\[ \frac{1}{2} \left( \left| F_1 \cap G_p \right| + \left| F_k \cap G_l \right| + \left| F_m \cap G_n \right| \right) \] (19)

which can be computed in time O(1), if the referenced
matrices have been precomputed. Thus we can compute
all shared directed butterflies in total time O(n^2). Dividing
by two, we get the number of shared undirected
butterflies.

Counting different butterfly topologies
Counting the number of different butterflies in the two
trees is done similar to counting the number of shared
butterflies. As before, we consider a pair of inner
edges, e ∈ T and e' ∈ T'. The quartets claimed by both e
and e', but with different butterfly topology, are on
the form a ∈ F_i ∩ G_p, b ∈ F_j ∩ G_q, c ∈ F_k ∩ G_l, and
d ∈ F_m ∩ G_n, for some claims F_i → (F_k, F_m) and
G_p → (G_l, G_n). The number of butterflies claimed by
both e and e' but with different topology is therefore
given by

\[ \frac{1}{2} \left( C''''[i, j] - R''''[i, j] \right) \] (20)

\[ R''''[i, j] = R''''[i, j] \] (21)

\[ C''''[i, j] - L''''[i, j] (R''''[i, j] - L''''[i, j]) \] (22)

Notice, however, that the inner sum is simply the total
sum of entries in I, M, except for the rows i and k
and columns j and l, see Figure 6(a). We divide by
two since we count each quartet twice, due to symmetry
between the (k, l) and (m, n) pairs.

The number of butterflies claimed by
both e and e' but with different topology is therefore
given by

\[ \frac{1}{2} \left( C''''[i, j] - R''''[i, j] \right) \] (20)

\[ R''''[i, j] = R''''[i, j] \] (21)

\[ C''''[i, j] - L''''[i, j] (R''''[i, j] - L''''[i, j]) \] (22)

which can be computed in time O(1), if the referenced
matrices have been precomputed. Thus we can compute
all shared directed butterflies in total time O(n^2). Dividing
by two, we get the number of shared undirected
butterflies.
depending on whether we have precomputed \( I''_d \) and \( l''_d \), or \( l''_e \) and \( l''_e \). We can thus compute Eq. (22) in time \( O(1) \) for each pair of inner edges \( e \in T \) and \( e' \in T' \) giving a total time of \( O(n^2) \) to compute different directed, and thus different undirected, butterfly topologies in the two trees.

To get the actual number of different butterflies we have to divide by four.

**Time analysis**

The running time of the algorithm is dominated by the time \( O \left( \min \left( \max \left( d_v, d_{v'} \right)^2, 2.376, d_v^2 d_{v'}, d_v d_{v'}^2 \right) \right) \) it takes to compute either \( I''_d \) and \( l''_d \), or \( l''_e \) and \( l''_e \), for each pair of nodes \( v \in T \) and \( v' \in T' \). Let \( O(n^{2+\alpha}) \) be the time it takes to multiply two \( n \times n \) matrices. In section 4 of Additional file 1 we show that the running of our algorithm is \( O(n^{2+\alpha}) \), where \( \alpha = \frac{\omega - 1}{2} \). Using the Coppersmith-Winograd algorithm [12] for matrix multiplication, where \( \omega = 2.376 \), this yields a running time of \( O(n^{2.688}) \).

**Results**

We have implemented our new algorithm and, for comparison, the \( O(n^3) \) and \( O(n^4) \) algorithms [9] for general trees. We chose those algorithm instead of those from [10,11], because the running time of those
algorithms are dependent on the degree of the nodes, while a major feature of our new algorithm is that it has a good asymptotical running time independent of the degree of the nodes. For matrix multiplication we link to a BLAS library, and expect that to choose the most efficient algorithm for matrix multiplication. In our experiments the vecLib library from Mac OS X is used. We have run benchmarks with trees with ten leaves up to trees with almost 15, 000 leaves. For each size, trees were generated in four different ways: general trees, binary trees, star trees and trees with one node of degree \( \frac{n}{2} \) surrounded by degree 3 nodes. The code that generated the trees is available in Additional file 2. For each of the ten possible combinations of topologies, one pair of trees were randomly generated, and the time used for the computation of the quartet distance was measured and plotted. Our experiments were run on a Mac-Pro with two Intel quad-core Xeon processors running at 2.26 GHz and with 8 GB RAM. As seen in Figure 8 the implementation of our new algorithm is significantly faster than the implementations of the competing algorithms, on trees with many leaves. In the worst cases our algorithm approaches \( O(n^3) \) which is expected if the BLAS implementation uses the \( O(n^3) \) matrix multiplication algorithm. Indeed Figure 9 shows that the slowest of our runs are on two star-shaped trees, where we need to multiply two \( n \times n \) matrices and where the time-complexity of the matrix multiplication algorithm is most important. However, in most cases our algorithm seems to be close to quadratic execution time, even though it apparently uses an asymptotically slow matrix multiplication algorithm.

**Conclusion**

We have derived, implemented and tested a new algorithm for computing the quartet distance. In theory our algorithm has execution time \( O(n^{\alpha+2}) \), where \( \alpha = \frac{\omega - 1}{2} \). With current knowledge of matrix multiplication this is \( O(n^{2.688}) \). If an algorithm for matrix multiplication in time \( O(n^2) \) is found this would make our algorithm run in time \( O(n^{2+\frac{1}{2}}) \). Experiments on our implementation shows it to be fast in practice, and that it can have a running time significantly better than the theoretical upper bound, depending on the topology of the trees being compared.

---

**Figure 8 Comparison of algorithms**

Plotted are our new algorithm compared to previously known \( O(n^1) \) and \( O(n^2) \) algorithms. In a log-log plot \( x^b \) becomes a straight line with the slope determined by \( b \). The lines in the plot are not regression lines, but are inserted to help the reader judge the time complexity of our implementations.
The line in the plot is not a regression line, but is inserted to help the reader judge the time complexity of our implementation.

**Additional material**

Additional file 1: Supplementary material containing mathematical derivations that are too tedious for the main text

Additional file 2: The python script used to generate the random trees for the experiments.

**Acknowledgements**

We are grateful to Chris Christiansen, Martin Randers and Martin S. Stissing for many fruitful discussions about the quartet distance.

**Author details**

1. Bioinformatics Research Centre (BIRC), Aarhus University, C. F. Møllers Alle 8, DK-8000 Aarhus C, Denmark. 2. Department of Computer Science, Aarhus University, Åbogade 34, DK-8200 Aarhus N, Denmark.

**Authors’ contributions**

JN, TM and CP developed the algorithm. AK implemented the algorithm. AK and JN benchmarked and evaluated the algorithm. JN, TM and CP wrote the paper. All authors read and approved the paper.

**Competing interests**

The authors declare that they have no competing interests.

**References**

1. Robinson DF, Foulds LR: Comparison of weighted labelled trees. Combinatorial mathematics, VI (Proc. 6th Austral. Comb.), Lecture Notes in Mathematics, Springer 1979, 119-126.
2. Waterman MS, Smith TF: On the similarity of dendrograms. Journal of Theoretical Biology 1978, 73:789-800.
3. Allen BL, Steel M: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics 2001, 5:1-13.
4. Robinson DF, Foulds LR: Comparison of phylogenetic trees. Mathematical Biosciences 1981, 53:131-147.
5. Estabrook G, McMorris F, Meacham C: Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. Syst Zool 1985, 34:193-200.
6. Steel M, Penny D: Distribution of tree comparison metrics-some new results. Syst Biol 1993, 42(2):126-141.
7. 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.
8. Brodal GS, Fagerberg R, Pedersen CNS: Computing the Quartet Distance Between Evolutionary Trees in Time O(n log n). Algorithmica 2003, 38:377-395.
9. Christiansen C, Mailund T, Pedersen CNS, Randers M: Algorithms for Computing the Quartet Distance between Trees of Arbitrary Degree. Proc. of Workshop on Algorithms in Bioinformatics (WABI), Volume 3692 of Lecture Notes in Bioinformatics (LNBI), Springer-Verlag 2005, 77-88.
10. Christiansen C, Mailund T, Pedersen CNS, Randers M, Stissing MS: Fast calculation of the quartet distance between trees of arbitrary degrees. Algorithms for Molecular Biology 2006, 1.
11. Stissing M, Pedersen CNS, Mailund T, Brodal GS, Fagerberg R: Computing the quartet distance between evolutionary trees of bounded degree. In Proceedings of the 5th Asia-Pacific Bioinformatics Conference 2007, Volume 5 of Series on Advances in Bioinformatics and Computational Biology Edited by: Sankoff D, Wang L, Chin F 2007, 101-110.
Coppersmith D, Winograd S: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 1990, 9:251-281.

doi:10.1186/1748-7188-6-15

Cite this article as: Nielsen et al: A sub-cubic time algorithm for computing the quartet distance between two general trees. Algorithms for Molecular Biology 2011 6:15.