Computing the Quartet Distance Between Trees of Arbitrary Degree

Chris Christiansen, Thomas Mailund, Christian N.S. Pedersen, and Martin Randers

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 [6,2] and refined Buneman [9,4], 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.
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 butterfly quartets, while topology (d): $a \times d$, is a 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 that computes the quartet distance in time $O(n \log n)$. In this paper, we present two algorithms that compute 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 $O(n^2d^2)$ 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. In Sect. 4 we draw our conclusions.

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.
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 algorithm, 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.

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 $T_{\text{rest}}$, 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 $a\times b\times c$. 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 $i$th 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 $O(kn^4)$ and space $O(kn)$.

### 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 e.g. [5]. 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 $T_{\text{rest}}$ gives a star quartet in $T$, and any leaf $x'$ in $T'_{\text{rest}}$ 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 $T_{\text{rest}}$ and $T'_{\text{rest}}$ 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 $T'_{\text{rest}}$ is either in $T_a$, $T_b$, $T_c$ or $T_{\text{rest}}$, 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 $T'_{\text{rest}}$. The same applies for elements in $T_b$ and $T_c$. This gives the equations:

\[
\begin{align*}
|T_a \cap T'_{\text{rest}}| &= |T_a| - (|T_a \cap T'_a| + |T_a \cap T'_b| + |T_a \cap T'_c|), \\
|T_b \cap T'_{\text{rest}}| &= |T_b| - (|T_b \cap T'_a| + |T_b \cap T'_b| + |T_b \cap T'_c|), \\
|T_c \cap T'_{\text{rest}}| &= |T_c| - (|T_c \cap T'_a| + |T_c \cap T'_b| + |T_c \cap T'_c|).
\end{align*}
\]

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 path, see Fig. 4 for an illustration.

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 $i$th 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: Case 1: The quartet has a star topology in both trees (shared quartet). Case 2: The quartet has an equal butterfly topology in both trees (shared quartet). Case 3: The quartet has a different butterfly topology in the trees. Case 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:

$$qdist(T, T') = Q_3 + Q_4 = BQ + BQ' - 2Q_2 - Q_3$$

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: The number of quartets with the same butterfly topology in two instances of the same tree is 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.

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 $T_1 \Leftarrow e_1 (T_2, T_3)$, 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.
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, \( cl = T_1 \rightarrow (T_2, T_3) \) and \( cl' = T'_1 \rightarrow (T'_2, T'_3) \) in \( T \) and \( T' \), respectively. The number of oriented butterfly quartets shared by the claims can be computed by the expression:

\[
\text{count}(cl, cl') = \left(\frac{|T_1 \cap T'_1|}{2}\right) \cdot \left(|T_2 \cap T'_2| \cdot |T_3 \cap T'_3| + |T_2 \cap T'_3| \cdot |T_3 \cap T'_2|\right) \tag{1}
\]

and the total number of shared oriented butterfly quartets between \( T \) and \( T' \) is obtained by summing over all pairs of claims from the two trees:

\[
2Q_2 = \sum_{cl \in T} \sum_{cl' \in T'} \text{count}(cl, cl') \tag{2}
\]

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 \( \left(\frac{d-1}{2}\right) \) 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.

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.
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 $ab \mid cd$ 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 --→∗ denote the reflexive and transitive closure of newly introduced edges, that is, for $e_1 --→∗ e_2$, $e_2$ 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 $e_1 --→∗ e_2$ 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.

The following propositions establish 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.

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

(3)

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

Proof. Consider the quartet $ab \rightarrow cd$ 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 $ab \rightarrow cd$. 
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 \( e'' \rightarrow e' \). 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 \( e \rightarrow e' \) 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 \( e'' \rightarrow e' \) 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 \( \text{count}(ecd, e'cd') \) in constant time, so to achieve the desired complexity for computing \( \mathcal{R} \) 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 \( e_1 \rightarrow e_2 \) 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 expanded 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), which counts the number of butterfly quartets that have different topologies instead of the ones with the same topology. Similar to (1), it can be calculated in constant time.

Observe that when comparing two extended claims \( \text{ecl} \) and \( \text{ecl}' \), 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:

\[
d\text{count}(\text{ecl}, \text{ecl}') = |T_1 \cap T_1'| \cdot |T_1 \cap T_3'| \cdot |T_2 \cap T_2'| \cdot |T_3 \cap T_1'| + \\
|T_1 \cap T_1'| \cdot |T_1 \cap T_2'| \cdot |T_2 \cap T_3'| \cdot |T_3 \cap T_1'| + \\
|T_1 \cap T_1'| \cdot |T_1 \cap T_2'| \cdot |T_2 \cap T_1'| \cdot |T_3 \cap T_2'| + \\
|T_1 \cap T_1'| \cdot |T_1 \cap T_3'| \cdot |T_2 \cap T_1'| \cdot |T_3 \cap T_2'|
\]

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

\[
4Q_3 = \sum_{\text{ecl} \in ET} \sum_{\text{ecl}' \in ET'} d\text{count}(\text{ecl}, \text{ecl}')
\]

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 above algorithms in Java (the implementation is available upon request) and performed a set of experiments to validate the theoretical time complexities of our algorithms. The performance of the \( O(n^2d^2) \) time algorithm has been verified by running the algorithm on trees where all nodes have the same degree \( d \). To verify the running time with respect to \( d \), we have run the algorithm for various degrees \( d \) but fixed numbers of leaves \( n \), . Similarly, to verify the running time with respect to \( n \), we have kept \( d \) constant while changing \( n \).

Fig. 10 shows the running time of the \( O(n^2d^2) \) time algorithm as a function of \( d \). For fixed \( n \), the running time evolves as \( O(d^2) \), which verifies the theoretical time complexity with respect to \( d \). Fig. 11 shows the running time of the algorithms as a function of the number of leaves \( n \). For fixed \( d \) the running time evolves as \( O(n^2) \) and \( O(n^3) \) respectively, which agrees with the theoretical time complexities. As expected the \( O(n^3) \) does not depend on \( d \), however for \( d = 3 \) the algorithm is slightly slower than for larger values of \( d \). 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.
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 $n$.

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.

Fig. 11 also show that the choice of algorithm for computing the quartet distance should depend on both $n$ and $d$ as expected. Note that all trees in our testing scenario have many internal nodes with a high degree, whereas trees generated using e.g. the Buneman [6, 7] and refined Buneman [8, 9] methods (which might reconstruct non-binary trees) usually contain only a few nodes of high degree. On such trees, the $O(n^2d^2)$ time algorithm can be expected to run faster than in our tests. More experiments are needed to investigate the performance of the algorithms on such trees. One can view the graphs in Fig. 11 as a worst case guideline of when to use the $O(n^3)$ time algorithm. Fig. 12 shows a direct comparison of the running times of the two algorithms. Since Fig. 11(b)
shows no significant variance in the running time for \( d \geq 20 \), Fig. [12] includes only two different plots for the \( O(n^3) \) algorithm.

4 Conclusions

The contributions of this paper are two algorithms for computing the quartet distance between trees of arbitrary degrees. Earlier algorithms focus on binary trees and do not work on general trees. In this paper we have also shown that the quartet distance between two general trees can be expressed in terms not involving star quartets. 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 at removing one or both factors of \( d \). Another approach for reducing the time complexity is to try to adapt the ideas from the \( O(n \log n) \) time algorithm in [3] to trees of arbitrary degree.

References