Information and Computation 294 (2023) 105082

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Building a small and informative phylogenetic supertree ☆

Jesper Jansson [a,b,*], Konstantinos Mampentzidis [c], Sandhya T.P. [a,d]

[a] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[b] Graduate School of Informatics, Kyoto University, Kyoto, Japan
[c] Department of Computer Science, Aarhus University, Aarhus, Denmark
[d] Department of Mathematics, Stockholm University, Stockholm, Sweden

ARTICLE INFO

Article history:
Received 1 August 2019
Received in revised form 30 August 2022
Accepted 3 August 2023
Available online 22 August 2023

Keywords:
Phylogenetic tree
Supertree
Rooted triplet
Consistency
Approximation algorithm

ABSTRACT

We combine two fundamental optimization problems related to the construction of phylogenetic trees called *maximum rooted triplets consistency* and *minimally resolved supertree* into a new problem, which we call *q-maximum rooted triplets consistency* (q-MAXRTC). It takes as input a set $\mathcal{R}$ of rooted, binary phylogenetic trees with three leaves each and asks for a phylogenetic tree with exactly q internal nodes that contains the largest possible number of trees from $\mathcal{R}$. We prove that q-MAXRTC is NP-hard to approximate within a constant, develop polynomial-time approximation algorithms for different values of q, and show experimentally that representing a phylogenetic tree by one having much fewer nodes typically does not destroy too much branching information. To demonstrate the algorithmic advantage of using trees with few internal nodes, we also propose a new algorithm for computing the *rooted triplet distance* that is faster than the existing algorithms when restricted to such trees.

1. Introduction

1.1. Background

Phylogenetic trees are used in biology to represent evolutionary relationships [14]. The leaves in such a tree correspond to species that exist today and internal nodes to ancestor species that existed in the past. Phylogenetic trees are also used in linguistics to deduce and visualize evolutionary relationships between natural languages [28].

An important problem when studying the evolution of species is, given some data describing the species, to construct a phylogenetic tree that supports the input data as much as possible. The supertree approach [6,13,17] deals with the challenging problem of constructing a reliable phylogenetic tree for a large set of species by combining several accurate trees for small, overlapping subsets of the species into one final tree called a *supertree*. Depending on the type of data that is available and the type of tree that should be constructed, one obtains several variants of the problem. In particular, many special cases where all of the input phylogenetic trees are required to have exactly the same set of leaf labels have been investigated in detail since the early 1970s [1]; in such problems, the output supertree is referred to as a *consensus tree* (see,

☆ A preliminary version of this article appeared in *Proceedings of the Nineteenth International Workshop on Algorithms in Bioinformatics* (WABI 2019), LIPIcs, Vol. 143, Article No. 1, pp. 1:1–1:14, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.

* Corresponding author.
E-mail addresses: jj@i.kyoto-u.ac.jp (J. Jansson), kmampent@cs.au.dk (K. Mampentzidis), thekkumpadan@math.su.se (T.P. Sandhya).

https://doi.org/10.1016/j.ic.2023.105082
0890-5401/

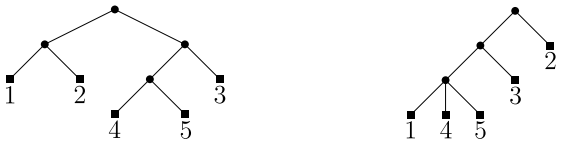


Fig. 1. Let $L = \{1, 2, 3, 4, 5\}$ and $\mathcal{R} = \{12|3, 24|1, 34|2, 35|2, 45|3\}$. In this example, no tree T such that $|\mathcal{R} \cap rt(T)| = 5$ exists. Left figure: optimal solution for MAXRTC with value 4. Right figure: optimal solution for 3-MAXRTC with value 3.

e.g., [10]). This article considers a new problem variant in which the input trees do not have identical leaf label sets, but the output supertree must have exactly q internal nodes for some fixed integer q.

1.2. Problem definition

A *rooted phylogenetic tree* is a rooted, unordered tree in which every leaf has a unique label and every internal node has at least two children. A *resolved triplet* is a binary rooted phylogenetic tree with three leaves. In this article, for simplicity we use the word "tree" to refer to a "rooted phylogenetic tree" and "triplet" to refer to a "resolved triplet". Also, we identify each leaf in a tree with its leaf label so that, e.g., a leaf labeled by x will be referred to by x. The triplet with leaves x, y, and z in which z is located the closest to the root is denoted by $xy|z$.

Let T be a tree. For any node u in T, $deg(u)$ is the number of children of u and $T(u)$ is the subtree induced by u and all proper descendants of u. For any two nodes u and v in T, $lca(u, v)$ is the lowest common ancestor node of u and v in T. A triplet $xy|z$ is said to be *induced by* T or *consistent with* T if x, y, and z are three leaves in T and $lca(x, z) = lca(y, z)$ and $lca(x, y) \neq lca(x, z)$ hold in T. Let $rt(T)$ be the set of all triplets induced by T. For any set $\mathcal{R}$ of triplets, if $\mathcal{R} \subseteq rt(T)$ then we say that $\mathcal{R}$ *is consistent with* T, or equivalently, that T *is consistent with* $\mathcal{R}$.

In the *q-maximum rooted triplets consistency problem*, abbreviated q-MAXRTC, the input is a set $\mathcal{R}$ of triplets over a leaf label set L and the objective is to find a tree T leaf-labeled by L containing exactly q internal nodes such that the value of $|\mathcal{R} \cap rt(T)|$ is maximized, i.e., the number of triplets induced by T that also belong to $\mathcal{R}$ must be as large as possible. An example of an instance of q-MAXRTC can be seen in Fig. 1. Throughout this article, we will denote the cardinality of the leaf label set L of the input $\mathcal{R}$ by n.

Let A be any algorithm for a maximization problem. Given an input instance I, let $opt(I)$ be the value of an optimal solution and $A(I)$ the value of the solution returned by A. Let $0 \leq r \leq 1$. We say that A is an r-approximation algorithm with *relative ratio* r, if $A(I) \geq r \cdot opt(I)$ for any I. Similarly, A is an r-approximation algorithm with *absolute ratio* r, if $A(I) \geq r \cdot |I|$ for any I. In particular, for q-MAXRTC we have that $A(I) \geq r \cdot |\mathcal{R}|$. From here on and unless otherwise stated, when we refer to any ratio r, we imply an absolute ratio.

1.3. Previous work

Aho *et al.* [2] proposed a polynomial-time algorithm, called BUILD, that can determine if there exists a tree inducing all triplets from an input $\mathcal{R}$, and if such a tree exists, output it. As observed by Bryant [9], the BUILD algorithm does not always produce a tree with the minimal number of internal nodes. In fact, BUILD might return a tree with $\Omega(n)$ more internal nodes than needed [19], which is undesirable because unnecessary internal nodes may suggest false groupings of the leaves, also known as *spurious novel clades* [6]. Moreover, scientists typically look for the simplest possible explanation for some given observations and would in many cases prefer a tree that makes as few additional statements as possible about evolutionary relationships that are not supported by the input data (for example, to construct a supertree for stony corals, Kerr [24] combined trees from 28 published studies, and when multiple trees were provided in a study without any clear preference, he would choose the least resolved one on the assumption that it gave "the most conservative estimates of relationship"). This motivates the *minimally resolved phylogenetic supertree* (MINRS) problem, where the output is a tree (if one exists) inducing all triplets from $\mathcal{R}$ while having the minimum number of internal nodes. The decision version of MINRS is NP-complete for $q \geq 4$ and polynomial-time solvable for $q \leq 3$ [19], where q is the total number of internal nodes allowed in the output tree. An exact exponential-time algorithm for MINRS and experimental results for the non-optimality of BUILD for MINRS were given in [22]. For the special case of *caterpillar trees* (trees in which every internal node has at most one non-leaf child), MINRS is polynomial-time solvable [19].

The above problems only consider finding trees that induce all triplets from $\mathcal{R}$. However, in situations where such a tree cannot be constructed, e.g., due to a single error in the input triplets, it is still useful to build a tree that induces as many of the triplets from $\mathcal{R}$ as possible. This has been formalized as the *maximum rooted triplets consistency problem* (MAXRTC). Bryant [9] showed that MAXRTC is NP-hard and Gąsieniec *et al.* [15] proposed a polynomial-time top-down $\frac{1}{3}$-approximation algorithm that always returns a caterpillar tree. Byrka *et al.* [12] showed that a bottom-up algorithm by Wu [32] can be modified to also obtain a polynomial-time $\frac{1}{3}$-approximation algorithm. In [11], Byrka *et al.* gave a polynomial-time $\frac{1}{3}$-approximation algorithm by derandomizing a randomized algorithm. In Section 4 below, we refer to the algorithm in [15] as *One-Leaf-Split* (`OLS`) and the algorithm in [12] as *the modified Wu algorithm* (`Mod-Wu`). For more results related to the computational complexity of MAXRTC, see [12].

Table 1
Previous and new results for computing q-MAXRTC. The abbreviations "abs." and "rel." correspond to "absolute" and "relative" respectively.

Year	Reference	Deterministic	q	Approximation	Type
1999	Gąsieniec *et al.* [15]	yes	unbounded	1/3	abs.
2010	Byrka *et al.* [11,12]	yes	$n-1$	1/3	abs.
2022	new [Section 3.1]	no	2	1/2	rel.
2022	new [Section 3.1]	yes	2	1/4	rel.
2022	new [Theorem 3.1]	yes	2	4/27	abs.
2022	new [Theorem 3.3]	yes	$q \geq 3$	$1/3 - 4/(3(q+q \bmod 2)^2)$	abs.

1.4. Motivation

The existing approximation algorithms for MAXRTC generally produce trees with an arbitrary number of internal nodes. For example, the algorithms in [11,12] always produce a tree with $n-1$ internal nodes, and the algorithm in [15] produces a tree with $n-1$ internal nodes for certain $\mathcal{R}$. However, due to the issue of spurious novel clades [6] mentioned above, biologists may prefer to build a supertree with few internal nodes that is still consistent with a large number of input triplets, which leads to the new problem q-MAXRTC introduced in this article. More precisely, q-MAXRTC can be regarded as a combination of MINRS and MAXRTC that models how well the triplet branching information contained in the set of input triplets can be preserved while forcing the size of the output tree to be bounded by a user-specified parameter q. Note that by the problem definition, some input branching structure usually has to be discarded and the objective is to do it in a least destructive way.

On a high level, q-MAXRTC is comparable to the problem of compressing a large data file into a small data file. As an analogy, consider the widely used JPEG compression method for images. Both JPEG and q-MAXRTC are examples of *lossy compression* where the user controls a parameter yielding a trade-off between the size of the compressed data (the number of bits for JPEG and the number of internal nodes for q-MAXRTC) and the amount of preserved information (the image quality for JPEG and the number of induced triplets in $\mathcal{R}$ for q-MAXRTC).

Finally, in the design of phylogenetic tree comparison algorithms, trees with fewer internal nodes sometimes admit faster running times. For example, given two trees built on the same leaf label set of size n, the fastest known algorithms for computing the so-called *rooted triplet distance* between the two trees takes $O(n \log n)$ time [7,8], but if at least one of the input trees has $O(1)$ internal nodes then the time complexity can be reduced to $O(n)$; see Section 5. As the available published trees get larger and larger (the total number of species on Earth was recently estimated to potentially be as many as one trillion [26]), to make their comparison practical, it may be necessary to approximate them using trees with fewer internal nodes while keeping enough triplet branching structure to represent them accurately.

1.5. New results and outline of the article

In Section 2, we show that q-MAXRTC is NP-hard for every fixed $q \geq 2$ and give some inapproximability results. Section 3 describes our approximation algorithms. In Section 4, we provide implementations, and present some experimental results showing that representing a tree by one having much fewer nodes typically does not destroy too much triplet branching information. As an extreme example, we show that allowing only nine internal nodes is still sufficient to capture on average 80% of the rooted triplets from some recently published trees, each having between 760 and 3081 internal nodes. Section 5 presents an $O(qn)$-time algorithm for computing the rooted triplet distance between two trees, where n is the size of the leaf label set in the two trees and q the number of internal nodes in the smaller tree, and provides an implementation of it. Finally, Section 6 contains some concluding remarks and open problems. For a summary of previous and new results related to q-MAXRTC, refer to Table 1.

2. Computational complexity of q-MAXRTC

In this section, we study the computational complexity of q-MAXRTC. We first establish the NP-hardness of q-MAXRTC, and then present some inapproximability results.

Theorem 2.1. *q-MAXRTC is NP-hard for every fixed $q \geq 2$.*

Proof. Consider the known NP-hard problem MAX q-CUT [23], in which the input is an undirected graph $G = (V, E)$ and the goal is to find a partition $(A_1, A_2, \ldots, A_q)$ of V such that the total number of edges connecting two vertices residing in different sets, i.e., *the size of the cut*, is maximized. We prove that q-MAXRTC is NP-hard by reducing MAX q-CUT to q-MAXRTC as follows: Let $L = V \cup \{z\}$ and $\mathcal{R} = \{xz|y,\ yz|x : \{x, y\} \in E\}$. We claim that there exists a cut $(A_1, A_2, \ldots, A_q)$ of size k in G if and only if there exists a solution to q-MAXRTC that is consistent with k triplets from $\mathcal{R}$. We now prove the claim.

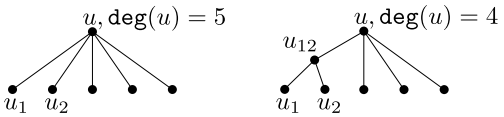


Fig. 2. Increasing the number of internal nodes by one without destroying any triplets.

First, assume that there exists a cut $(A_1, A_2, \ldots, A_q)$ of size k in G. We construct a tree T that is rooted at a node a_1 with additional internal nodes $a_2, \ldots, a_q$ such that a_i is a child of a_{i-1} for $2 \leq i \leq q$. For $i \in \{1, 2, \ldots, q\}$, $|A_i|$ leaves bijectively labeled by A_i are attached as children of a_i, and the leaf z is added as a child of a_q. Consider any edge $\{x, y\}$ in the cut. By the definition of a cut, $x \in A_i$ and $y \in A_j$ for two different $i, j \in \{1, 2, \ldots, q\}$. If $i < j$, then $yz|x$ will be consistent with T, since $lca(y, z) = a_j$ is a proper descendant of $lca(x, y) = lca(x, z) = a_i$. Similarly, if $i > j$, then $xz|y$ will be consistent with T. For every edge in the cut, exactly one triplet from $\mathcal{R}$ will be consistent with T, so in total, T will be consistent with exactly k triplets from $\mathcal{R}$.

Conversely, assume that there exists a tree T with q internal nodes $a_1, a_2, \ldots, a_q$ that is consistent with k triplets from $\mathcal{R}$. Define $A_i = \{x : x \text{ is a child of } a_i\} \setminus \{z, a_1, a_2, \ldots, a_q\}$ for $1 \leq i \leq q$ and let $S = \mathcal{R} \cap rt(T)$. For each $xz|y \in S$, clearly x and y belong to different sets A_i and A_j for some $i, j \in \{1, 2, \ldots, q\}$, and thus the corresponding edge $\{x, y\}$ contributes one to the size of the cut, making the size of the cut $|S| = k$. □

From the inapproximability of MAXCUT [16], we obtain the following corollary:

Corollary 2.1. *Unless P=NP, 2-MAXRTC cannot be approximated in polynomial time within a relative ratio of* $16/17 + \epsilon$*, for any constant* $\epsilon > 0$.

From the inapproximability of MAX q-CUT [23], we obtain the following corollary:

Corollary 2.2. *Unless P=NP, for any* $q \geq 3$*, it holds that q-MAXRTC cannot be approximated in polynomial time within a relative ratio of* $1 - 1/(34q) + \epsilon$*, for any constant* $\epsilon > 0$.

Remark 1. Recall from Section 1 that MINRS is polynomial-time solvable if restricted to caterpillar trees [19]. In contrast, the proof of Theorem 2.1 shows that q-MAXRTC remains NP-hard even in this special case.

3. Approximability of q-MAXRTC

Intuitively, a tree with a larger number of internal nodes should be able to induce more triplets from a given $\mathcal{R}$. The next lemma shows that this is indeed so, and upper bounds the total number of triplets that can be induced. Define $opt(q)$ to be the maximum number of triplets from $\mathcal{R}$ that can be consistent with a tree T with q internal nodes.

Lemma 3.1. *For any integers* q, q' *satisfying* $2 \leq q' \leq q \leq n - 1$*, we have* $opt(q') \leq opt(q) \leq \lceil \frac{q-1}{q'-1} \rceil opt(q')$.

Proof. We start by showing that $opt(q') \leq opt(q)$. Let T' be a tree with q' internal nodes that induces $opt(q')$ triplets from $\mathcal{R}$. We can create a tree T with q internal nodes that induces at least as many triplets from $\mathcal{R}$ as follows.

- Let $T = T'$.
- While T does not have q internal nodes, repeat the following steps:
  - Select any node $u \in T$ with $deg(u) > 2$. (Such a node must exist because if every internal node only had two children then the number of internal nodes would be $n-1$, which contradicts the fact that T has fewer than $q \leq n-1$ internal nodes.)
  - Let u_1, u_2 be any two children of u.
  - Create an internal node u_{12}, and make u_1 and u_2 the children of u_{12} and u_{12} the child of u.

  An example of the construction can be found in Fig. 2.

To show the second part of the inequality, proceed as follows. Define the *delete operation* on any non-root node u in a tree as the operation of making the children of u become children of the parent of u, and then removing u and all edges incident to u. Let T be a tree with q internal nodes that induces $opt(q)$ triplets from $\mathcal{R}$ and let W be any subset of the set of internal nodes from T such that W includes the root of T. Suppose that we create a tree T' by letting T' be a copy of T and applying the delete operation to every internal node of T' not in W. See Fig. 3 for an example. Consider any triplet $ab|c$ induced by a tree to be anchored in the node $lca(a, b)$. Then, if $ab|c$ is anchored in a node u in T, where $u \in W$, it is also anchored in u in T' and u will still be a proper descendant of $lca(a, c)$ in T' because $lca(a, c)$ in T' lies somewhere on the

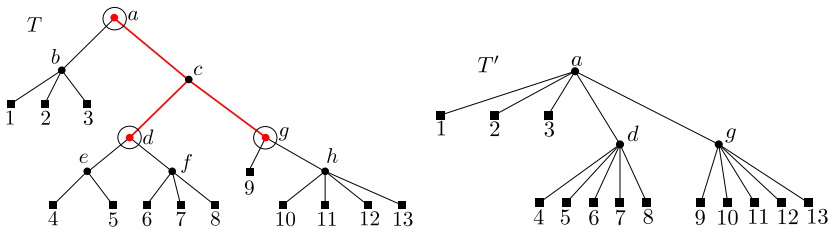


Fig. 3. Let T be the tree on the left with eight internal nodes and let $W = \{a, d, g\}$. The tree T' on the right with three internal nodes is created by deleting all internal nodes in T except the ones in W.

path from the root of T' to the parent of u in T'. Thus, for every node u in T such that $u \in W$, all triplets anchored in u will also be induced by T'.

Now, define $\lambda = \lceil \frac{q-1}{q'-1} \rceil$ and use the following procedure to build λ trees $T'_1, T'_2, \ldots, T'_\lambda$ like in the description of T' above, each having at most q' internal nodes and with the property that every internal node $u \in T$ except the root corresponds to an internal node of exactly one such tree:

- Store all internal nodes of T except the root in a list S in any order, and let $j = 1$.
- Initialize W to be a set containing the root of T only. If $|S| \geq q' - 1$, remove the first $q' - 1$ elements from S and insert them into W; otherwise, remove all remaining elements from S and insert them into W. Construct T'_j by taking a copy of T and applying the delete operation to every internal node not in W. Let $j = j + 1$.
- If $|S| = 0$, stop. Otherwise, go to the previous step.

We then have: $|rt(T) \cap \mathcal{R}| = opt(q) \leq \sum_{i=1}^{\lambda} |rt(T'_i) \cap \mathcal{R}| \leq \lambda \cdot opt(q') = \lceil \frac{q-1}{q'-1} \rceil \cdot opt(q')$. □

3.1. Polynomial-time approximation algorithms based on MAX 3-CSP

MAX 3-AND is a Boolean satisfiability problem in which we are given as input a logical formula consisting of a set of clauses, each being a conjunction (AND) of three literals formed from a set of Boolean variables, and the goal is to assign each Boolean variable a *True*/*False*-value so that the total number of satisfied clauses is maximized. Both MAX 3-AND and the well-known MAX 3-SAT problem are special cases of the MAX 3-CSP problem [33], where a clause can be an arbitrary function over three literals. The following lemma shows that 2-MAXRTC can be reduced to MAX 3-AND in polynomial time while preserving the approximation ratio.

Lemma 3.2. *If MAX 3-AND can be approximated within a factor of r, then 2-MAXRTC can also be approximated within a factor of r.*

Proof. We present a reduction from 2-MAXRTC to MAX 3-AND. Given a set of triplets $\mathcal{R}$ built on the leaf label set L, we construct an instance of MAX 3-AND as follows: let $V = L$ be the set of variables and $C = \{x \wedge y \wedge \bar{z} : xy|z \in \mathcal{R}\}$ the set of clauses. We claim that there exists a solution to MAX 3-AND that satisfies k clauses from C if and only if there exists a solution to 2-MAXRTC that is consistent with k triplets from $\mathcal{R}$. We now prove this claim.

First, suppose that there exists an assignment ϕ on V that satisfies a set S of clauses from C, where $|S| = k$. Let $A = \bigcup_{x \wedge y \wedge \bar{z} \in S} \{z\}$ and $B = \bigcup_{x \wedge y \wedge \bar{z} \in S} \{x, y\}$. Observe that $A \cap B = \emptyset$, which can be argued as follows. Take any $c \in A \cap B$. By the definition of A, $\bar{c}$ is *True* in ϕ. This means that $c =$ *False* in ϕ, while $c \in B$ implies that $c =$ *True* in ϕ, leading to a contradiction. Next, we construct a tree T with root a and an internal node b that is a child of a that is consistent with k triplets from $\mathcal{R}$. The elements in A and B are added as children of a and b respectively. The set of all k triplets of the form $\{xy|z : x \wedge y \wedge \bar{z} \in S\}$ is consistent with T, since $lca(x, y) = b$ and $lca(x, z) = lca(y, z) = a$.

Conversely, assume that T is a tree consistent with a set S' of k triplets from $\mathcal{R}$. Let a be the root of T and b the internal node of T that is a child of a. Let $B = \{c : c \text{ is a child of } b\}$ and $A = \{c : c \text{ is a child of } a\} \setminus \{b\}$. We assign $c =$ *True* for every $c \in B$ and $c =$ *False* for every $c \in A$. Then, for every $x, y \in B$ and $z \in A$, the corresponding clause $x \wedge y \wedge \bar{z}$ is satisfied. Hence, all clauses corresponding to the triplets in S' are satisfied. □

Lemma 3.2 allows every approximation algorithm for MAX 3-AND to be used to approximate 2-MAXRTC. For MAX 3-AND, Zwick [33] presented a randomized polynomial-time $\frac{1}{2}$-approximation algorithm with relative ratio based on semi definite programming. Trevisan [30] presented a deterministic polynomial-time $\frac{1}{4}$-approximation algorithm with relative ratio based on linear programming. A deterministic polynomial-time algorithm based on local search by Alimonti [3], would satisfy $\geq \frac{1}{8}|C|$ number of clauses, giving a $\frac{1}{8}$-approximation ratio for 2-MAXRTC. Since this ratio is absolute, from Lemma 3.1 this algorithm also gives a $\frac{1}{8}$-approximation ratio for q-MAXRTC, where $q \geq 3$.

3.2. Polynomial-time approximation algorithms based on derandomization

Reducing 2-MAXRTC to MAX 3-AND can produce a polynomial-time deterministic $\frac{1}{8}$-approximation algorithm for q-MAXRTC, however from Lemma 3.1, we should be able to capture more triplets by allowing more internal nodes. The algorithms based on MAX 3-AND cannot be directly extended to support Lemma 3.1. We propose a new deterministic polynomial-time algorithm for q-MAXRTC that achieves a $\frac{4}{27}$-approximation ratio, based on a randomized algorithm for 2-MAXRTC, and then show how to extend it to get better approximation ratios for higher values of q. Note that Byrka *et al.* previously used a similar derandomization technique in [11]; however, the time complexity of their method (Theorem 1 in [11]) is worse than Theorem 3.1 below since we have optimized the derandomization procedure for trees rather than the more complex *phylogenetic networks* studied in their article (for a definition, see Section 2 of [11]). Furthermore, when restricted to trees, the method of Byrka *et al.* only considers binary trees on n leaves and thus does not allow cases where $q < n-1$.

Lemma 3.3. *There exists a randomized polynomial-time $\frac{4}{27}$-approximation algorithm for q-MAXRTC.*

Proof. Let $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$ be the set of triplets and $L = \{x_1, \ldots, x_n\}$ the leaf label set. Build a tree T with two internal nodes, with a being the root and b the child of the root. Make every leaf $x_i \in L$ with probability $\frac{2}{3}$ a child of b and probability $\frac{1}{3}$ a child of a. Let Y_j be a random variable that is 1 if $r_j \in rt(T)$ and 0 otherwise. Let $W = \sum_{j=1}^{|\mathcal{R}|} Y_j$. For the expected number of triplets consistent with T we have $E[W] = \sum_{j=1}^{|\mathcal{R}|} E[Y_j] = \sum_{j=1}^{|\mathcal{R}|} \frac{4}{27} = \frac{4}{27}|\mathcal{R}|$. □

Theorem 3.1. *There exists a deterministic polynomial-time $\frac{4}{27}$-approximation algorithm for q-MAXRTC that runs in $O(|\mathcal{R}|)$ time.*

Proof. We derandomize the algorithm in Lemma 3.3 with the method of conditional expectations [31] in a way that differs from Byrka *et al.* [11], where the main focus is the general case of phylogenetic networks. In our method, the leaves $x_1, \ldots, x_n$ are scanned from left to right, and each leaf is deterministically assigned to either be the child of b, denoted $x_i \leftarrow b$, or the child of a, denoted $x_i \leftarrow a$. The leaves are assigned so that after every assignment, the expected value of the solution does not decrease. From probability theory we have $E[W] = \frac{1}{3}E[W|x_1 \leftarrow a] + \frac{2}{3}E[W|x_1 \leftarrow b]$. We choose $n_1 = a$ or $n_1 = b$ such that $E[W|x_1 \leftarrow n_1] = \max(E[W|x_1 \leftarrow a], E[W|x_1 \leftarrow b])$. Then $E[W|x_1 \leftarrow n_1] \geq E[W] = \frac{4}{27}|R|$. Suppose that the first i leaves have been assigned to $n_1, \ldots, n_i$. Let those assignments be contained in N_i, i.e., we have that $N_i = \{x_1 \leftarrow n_1, \ldots, x_i \leftarrow n_i\}$. To find the assignment for x_{i+1} we follow the same approach as that for x_1, i.e., we have $E[W|N_i] = \frac{1}{3}E[W|N_i, x_{i+1} \leftarrow a] + \frac{2}{3}E[W|N_i, x_{i+1} \leftarrow b]$ and then n_{i+1} is chosen so that $E[W|N_{i+1}] = \max(E[W|N_i, x_{i+1} \leftarrow a], E[W|N_i, x_{i+1} \leftarrow b])$. By induction, we then get that $E[W|N_{i+1}] \geq E[W|N_i] \geq \cdots \geq \frac{4}{27}|\mathcal{R}|$.

To compute $E[W|N_i]$, we use the fact that $E[W|N_i] = \sum_{j=1}^{|\mathcal{R}|} Pr[r_j \in rt(T)|N_i]$, where $Pr[r_j \in rt(T)|N_i]$ can be computed in O(1) time (see procedure PR2() in Algorithm 1). A direct implementation that scans all the leaves and for every possible assignment of a leaf x_i, computes the expected value $E[W|N_i]$ by scanning the entire set $\mathcal{R}$ (see procedure 2-MAXRTC-SLOW() in Algorithm 1) would require $O(n|\mathcal{R}|)$ time.

Algorithm 1 $O(n|\mathcal{R}|)$-time $\frac{4}{27}$-approximation algorithm for q-MAXRTC based on 2-MAXRTC (Theorem 3.1).

```
1: procedure PR2(xy|z)                                   ▷ Computing Pr[xy|z ∈ rt(T)|N_i]
2:     if x ← a or y ← a or z ← b then return 0
3:     p = 4/27
4:     if x ≠ ∅ and x ← b then p = 3p/2                  ▷ x ≠ ∅, thus x has been assigned
5:     if y ≠ ∅ and y ← b then p = 3p/2
6:     if z ≠ ∅ and z ← a then p = 3p
7:     return p

8: procedure 2-MAXRTC-SLOW(R)                            ▷ The main procedure
9:     for i = 1 to n do
10:        x_i ← ∅
11:    for i = 1 to n do
12:        aValue = 0                                    ▷ To compute E[W|N_{i-1}, x_i ← a]
13:        bValue = 0                                    ▷ To compute E[W|N_{i-1}, x_i ← b]
14:        for j = 1 to |R| do
15:            x_i ← a
16:            aValue = aValue + PR2(R[j])
17:            x_i ← b
18:            bValue = bValue + PR2(R[j])
19:        x_i ← b
20:        if aValue > bValue then x_i ← a
```

Algorithm 2 $O(|\mathcal{R}|)$ $\frac{4}{27}$-approximation algorithm for q-MAXRTC based on 2-MAXRTC.

```
1: procedure 2-MAXRTC-FAST(R)                                   ▷ The main procedure
2:     for i = 1 to n do
3:         x_i ← ∅
4:     prev = 4|R|/27                                           ▷ Storing E[W|N_0], where N_0 = ∅
5:     for i = 1 to n do
6:         aValue = prev                                        ▷ To compute E[W|N_{i-1}, x_i ← a]
7:         bValue = prev                                        ▷ To compute E[W|N_{i-1}, x_i ← b]
8:         for j = 1 to |F[x_i]| do
9:             x_i ← ∅
10:            aValue = aValue − PR2(F[x_i][j])
11:            bValue = bValue − PR2(F[x_i][j])
12:            x_i ← a
13:            aValue = aValue + PR2(F[x_i][j])
14:            x_i ← b
15:            bValue = bValue + PR2(F[x_i][j])
16:        x_i ← b
17:        prev = bValue
18:        if aValue > bValue then
19:            x_i ← a
20:            prev = aValue
```

We can achieve a more efficient implementation (see procedure 2-MAXRTC-FAST() in Algorithm 2) that runs in $O(|\mathcal{R}|)$ time, by maintaining for every leaf $x_i \in L$, a list of all the triplets from $\mathcal{R}$ that x_i is part of, denoted $F[x_i]$. At the beginning of the i-th iteration of the first for-loop in Algorithm 2, the value of the variable *prev* is $E[W|N_{i-1}]$. To determine the assignment for leaf x_i, we need to compute $E[W|N_{i-1}, x_i \leftarrow a]$ and $E[W|N_{i-1}, x_i \leftarrow b]$, and for this we use the second for-loop. At the end of the execution of the second for-loop, we have that the value of $E[W|N_{i-1}, x_i \leftarrow a]$ will be stored in the variable *aValue* and the value of $E[W|N_{i-1}, x_i \leftarrow b]$ in the variable *bValue*. To compute *aValue* (resp. *bValue*), we initialize it to the value of *prev*, and then for every triplet in the list $F[x_i]$, we subtract the contribution of that triplet to the value of *prev* when $x_i \leftarrow \emptyset$, and add its new contribution by having $x_i \leftarrow a$ (resp. $x_i \leftarrow b$). Since every triplet in $\mathcal{R}$ will be part of three lists, every triplet induces O(1) calls to the procedure PR2() in Algorithm 1, giving the $O(|\mathcal{R}|)$ final bound of 2-MAXRTC-FAST(). □

In the following theorem, we prove that the best possible absolute approximation ratio for 2-MAXRTC is $\frac{4}{27}$, making the approximation algorithm in Theorem 3.1 optimal when considering algorithms with absolute approximation ratios.

Theorem 3.2. *For any $\epsilon > 0$, there exists some n and set $\mathcal{R}$ of triplets on a leaf label set of size n, such that the approximation ratio $\geq \frac{4}{27} + \epsilon$ for 2-MAXRTC is impossible.*

Proof. For any n, let $L_n = \{1, 2, \ldots, n\}$ and $\mathcal{R}_n = \{ab|c, ac|b, bc|a : a, b, c \in L, |\{a, b, c\}| = 3\}$. Since $|L_n| = n$, we have $|\mathcal{R}_n| = 3\binom{n}{3}$. Construct a tree T with two internal nodes, which is rooted at the vertex a with an internal node b (i.e., b is a child of a). Let $B = \{c : c \text{ is a child of } b\}$ and $A = \{c : c \text{ is a child of } a\} \setminus \{b\}$. Assume that $m = |B|$. Then $|A| = n - m$ and $|rt(T) \cap \mathcal{R}_n| = m(\frac{m-1}{2})(n - m)$. By taking derivatives, we obtain that T is consistent with the largest number of triplets when $m = \frac{n+1+\sqrt{n^2-n+1}}{3}$. For that given m, we then have $|rt(T) \cap \mathcal{R}_n| = \left(\frac{n+1+\sqrt{n^2-n+1}}{3}\right)\left(\frac{n-2+\sqrt{n^2-n+1}}{6}\right)\left(\frac{2n-1-\sqrt{n^2-n+1}}{3}\right)$ and $\lim_{n\to\infty} \frac{|rt(T)\cap\mathcal{R}_n|}{|\mathcal{R}_n|} = \frac{4}{27}$. □

To obtain an algorithm with a better approximation ratio for $q \geq 3$ (Algorithm 3), we allow the output tree T to have q internal nodes $\{u_1, \ldots, u_q\}$. Every internal node $u_j \in T$ has a probability $p(u_j)$, which is the probability of a fixed leaf being assigned to that node. Given that $\sum_{j=1}^{q} p(u_j) = 1$, we can obtain a randomized algorithm, the analysis of which follows from Lemma 3.3. Let $E[W]$ be the expected value of that randomized algorithm. Like in Theorem 3.1, we can derandomize the algorithm to obtain a $\frac{E[W]}{|\mathcal{R}|}$-approximation ratio. The only difference in the proof is that the total number of possible assignments is q instead of 2, i.e., given N_i, we choose n_{i+1} for x_{i+1} such that $E[W|N_i, x_{i+1} \leftarrow n_{i+1}] = \max(E[W|N_i, x_{i+1} \leftarrow u_1], \ldots, E[W|N_i, x_{i+1} \leftarrow u_q])$. The problem is thus reduced to finding a tree with q internal nodes and a choice of probabilities $p(u_1), \ldots, p(u_q)$ such that $E[W] > \frac{4}{27}|\mathcal{R}|$.

Algorithm 3 The $O(q|\mathcal{R}|)$-time algorithm for q-MAXRTC when $q > 2$ (Theorem 3.3).

procedure PRQ($xy|z$, q, k) ▷ Computing $Pr[xy|z \in rt(T)|N_i]$
  if $x \leftarrow \emptyset$ **and** $y \leftarrow \emptyset$ **and** $z \leftarrow \emptyset$ **then** return $1/3 - 4/(3(q+1)^2)$
  if $x \leftarrow u$ **and** $y \leftarrow \emptyset$ **and** $z \leftarrow \emptyset$ **then** return $(u_{\uparrow d} + k)/(k+1)^2$
  if $x \leftarrow \emptyset$ **and** $y \leftarrow v$ **and** $z \leftarrow \emptyset$ **then** return $(v_{\uparrow d} + k)/(k+1)^2$
  if $x \leftarrow \emptyset$ **and** $y \leftarrow \emptyset$ **and** $z \leftarrow w$ **then** return $(k + 2w_{\uparrow s})/(k+1)^2$
  if $x \leftarrow u$ **and** $y \leftarrow v$ **and** $z \leftarrow \emptyset$ **then**
    $p = lca(u, v)$
    return $(k + 1 - p_{\downarrow})/(k+1)$
  if $x \leftarrow u$ **and** $y \leftarrow \emptyset$ **and** $z \leftarrow w$ **then**
    $p = lca(u, w)$
    if $p == u$ **then** return 0
    $m =$ `child of p whose subtree contains` u
    return $m_{\downarrow}/(k+1)$
  if $x \leftarrow \emptyset$ **and** $y \leftarrow v$ **and** $z \leftarrow w$ **then**
    $p = lca(v, w)$
    if $p == v$ **then** return 0
    $m =$ `child of p whose subtree contains` v
    return $m_{\downarrow}/(k+1)$
  if $x \leftarrow u$ **and** $y \leftarrow v$ **and** $z \leftarrow w$ **then**
    return $(lca(x, z) == lca(y, z)$ **and** $lca(x, y) \neq lca(x, z))$

procedure Q-MAXRTC($\mathcal{R}$, q) ▷ The main procedure
  if $q \mod 2 == 1$ **then**
    $q = q - 1$
  $k = q/2$
  $prev = |\mathcal{R}|(1/3 - 4/(3(q+1)^2))$ ▷ Storing $E[W|N_0]$, where $N_0 = \emptyset$
  for $i = 1$ `to` n **do**
    for $m = 1$ `to` $k+1$ **do**
      $nValue[m] = prev$
    for $j = 1$ `to` $|F[x_i]|$ **do**
      for $m = 1$ `to` $k+1$ **do**
        $x_i \leftarrow \emptyset$
        $nValue[m] = nValue[m] -$ PRQ($F[x_i][j]$)
        $x_i \leftarrow u_m$
        $nValue[m] = nValue[m] +$ PRQ($F[x_i][j]$)
    $x_i \leftarrow u_1$
    $curBest = nValue[1]$
    for $j = 2$ `to` $k+1$ **do**
      if $nValue[j] > curBest$ **then**
        $curBest = nValue[j]$
        $x_i \leftarrow u_j$
        $prev = curBest$

Theorem 3.3. *Given $q \geq 3$, there exists a randomized polynomial-time algorithm for q-MAXRTC that achieves a $(\frac{1}{3} - \frac{4}{3(q+q \bmod 2)^2})$-approximation. The algorithm can be derandomized while preserving the approximation ratio. The running time of the deterministic algorithm is $O(q|\mathcal{R}|)$.*

Proof. Let $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$ and $L = \{x_1, \ldots, x_n\}$. If $q = 2k+1$ for some $k \in \mathbb{N}$, build a binary tree T with q nodes in total. By construction, T has $k+1$ leaves and k internal nodes. Assign the probability $1/(k+1)$ to every leaf and 0 to every internal node. For the expected value of the randomized algorithm we then obtain:

$$E[W] = \sum_{j=1}^{|\mathcal{R}|} Pr[r_j \in rt(T)] = \sum_{j=1}^{|\mathcal{R}|} \frac{k(k+1) + 2\binom{k+1}{3}}{(k+1)^3} = \left(\frac{1}{3} - \frac{1}{3(k+1)^2}\right)|\mathcal{R}| \,.$$

By setting $k = \frac{q-1}{2}$, we get $E[W] = \left(\frac{1}{3} - \frac{4}{3(q+1)^2}\right)|\mathcal{R}|$. If $q = 2k$, we follow the same approach but by building a binary tree T with $2k-1$ nodes instead. The tree T has k leaves and $k-1$ internal nodes. Every leaf has the probability $1/k$ and every internal node the probability 0. After finding the assignment of the leaves according to the chosen probabilities, we recursively remove all nodes with degree 0 from the tree that are not leaves. (These nodes correspond to internal nodes that have not been assigned any leaves.) We then pick one internal node u for which $deg(u) \geq 3$ to add a new internal node as shown in Fig. 2. This would give $E[W] \geq \left(\frac{1}{3} - \frac{4}{3q^2}\right)|\mathcal{R}|$. Hence, for any $q \geq 3$ we have $E[W] \geq \left(\frac{1}{3} - \frac{4}{3(q+q \bmod 2)^2}\right)|\mathcal{R}|$.

To obtain a deterministic algorithm that preserves the approximation ratios, we use the method of conditional expectations. We now describe how to achieve the $O(q|\mathcal{R}|)$ time bound, given any binary tree T with q internal nodes and a probability assignment as described above. For any internal node u in T, let $d(u)$ be the number of edges on the path from the root of T to u. Define the following counters:

- $u_{\downarrow}$ = total number of leaves in $T(u)$.
- $u_{\uparrow s}$ = total number of pairs of leaves x, y such that $x \neq y$, x and y are not in $T(u)$, and $d(lca(u,x)) = d(lca(u,y))$.
- $u_{\uparrow d}$ = total number of pairs of leaves x, y such that $x \neq y$, x and y are not in $T(u)$, and $d(lca(u,x)) < d(lca(u,y))$.

After building T, all these counters can be trivially computed in $O(q)$ time by applying two depth first search traversals on T. The procedure PRQ() of Algorithm 3 shows how to compute $Pr[xy|z \in rt(T)|N_i]$. The most expensive operation is computing the lowest common ancestor of two nodes. However, data structures exist [4], that can answer such queries in $O(1)$ time, while only requiring $O(q)$ preprocessing time. Given such a data structure, the time complexity of PRQ() becomes $O(1)$. The main procedure Q-MAXRTC() of Algorithm 3 is a direct extension of the main procedure 2-MAXRTC-FAST() of Algorithm 2, with the only difference being that we have q possible assignments for a leaf instead of 2. Hence, every triplet in $\mathcal{R}$ induces $O(q)$ calls to PRQ() of Algorithm 3, making the entire complexity of the algorithm $O(q) + O(q|\mathcal{R}|) = O(q|\mathcal{R}|)$. □

4. Implementation and experiments

We used the C++ programming language to implement the algorithm from Theorem 3.1 for 2-MAXRTC, and the deterministic algorithm from Theorem 3.3 for q-MAXRTC when $q > 2$. The implementation is publicly available at https://github.com/kmampent/qMAXRTC. For simplicity in our implementation, given two nodes u and v, a straightforward $O(q)$-time algorithm is used to compute $lca(u, v)$. To evaluate our algorithms experimentally, we also implemented *One-Leaf-Split* (`OLS`) [15] and *the modified Wu algorithm* (`Mod-Wu`) [12]. Below, we describe some experiments on simulated and real datasets and the results.

4.1. Simulated datasets

To evaluate the algorithms, we simulate the process of trying to reconstruct an unknown phylogenetic tree with a specified number of internal nodes from an input set of triplets $\mathcal{R}$ that can be incomplete, i.e., missing some triplets, and that can contain incorrect triplets due to experimental errors. To cover various possible scenarios, we define the following types of sets for $\mathcal{R}$:

- *dense consistent* (abbreviated `dc`): $|\mathcal{R}| = \binom{n}{3}$ and $\mathcal{R}$ is consistent with a tree T containing $n-1$ internal nodes. The tree T is created using the *uniform model* (also known in the literature as the *PDA model*), which is a standard method for generating a random binary phylogenetic tree with properties that are similar to those of phylogenetic trees from the real world; see, e.g., [27] or p. 29 of [29] for details.
- *probabilistic*: $|\mathcal{R}| = n^2$ and $\mathcal{R}$ is a set of triplets on n leaf labels created as follows. Initially, $\mathcal{R} = \emptyset$. After building a binary tree T on n leaves following the uniform model [27,29], extract triplets at random from T and add them to $\mathcal{R}$. For each extracted triplet, permute its leaves uniformly at random with probability p. Depending on whether $p = 0.25$, $p = 0.50$, or $p = 0.75$, the abbreviations we use are `prob25`, `prob50`, and `prob75`, respectively.
- *noisy*: $|\mathcal{R}| = n^2$ and $\mathcal{R}$ is a set of triplets on n leaf labels that is created at random.

In the experiments of this dataset, the *performance* of an algorithm for any fixed q, n, and dataset model is defined as its mean approximation ratio, taken over 100 randomly generated instances of size n. Fig. 4 compares the performance of q-MAXRTC, `OLS`, and `Mod-Wu` in the `dc`, `prob25`, `prob50`, `prob75`, and noisy models defined above for small values of q and n at most 1000. In all models, the larger the value of q, the better the performance of q-MAXRTC. Moreover, the rate of improvement in performance decreases as the value of q increases, which is expected.

When comparing 9-MAXRTC against `OLS` & `Mod-Wu`, one can see in Fig. 4 that while `OLS` & `Mod-Wu` perform better, the difference in performance is small when there are few conflicting triplets like in the `dc` and `prob25` models. All methods' relative performance decrease as the probability p increases.

For the `dc` dataset, which contains no conflicting triplets, the performance of 9-MAXRTC is in some cases even better than that of `Mod-Wu` according to Fig. 4. Significantly, a tree with just nine internal nodes obtained by setting $q = 9$ can induce close to 80% of the triplets even if the input tree contains as many as 1000 leaves. Also note that an optimal tree with nine internal nodes might in fact induce even more than 80% of the triplets here since the algorithm from Theorem 3.3 that was used in the experiments is an approximation algorithm.

Fig. 5 shows the performance of q-MAXRTC for $q \in \{2, 5, 7, 9\}$ in every simulation model. In all experiments, the larger the value of q, the better the performance. As expected, the performance deteriorates when the number of conflicting triplets in $\mathcal{R}$ is increased.

4.2. Real datasets

We considered five trees from recently published articles ([18] and [25]). From [18] we used the trees from the supplementary datasets 2 and 4, denoted `nmS2` and `nmS4` respectively. From [25] we used the trees from the supplementary datasets 1, 2, and 4, denoted `poS1`, `poS2`, and `poS4` respectively. All trees are binary except `nmS2` and `nmS4`. However, we removed the leaf that is a child of `nmS2`'s root to make `nmS2` binary. Similarly, we removed the two leaves that are

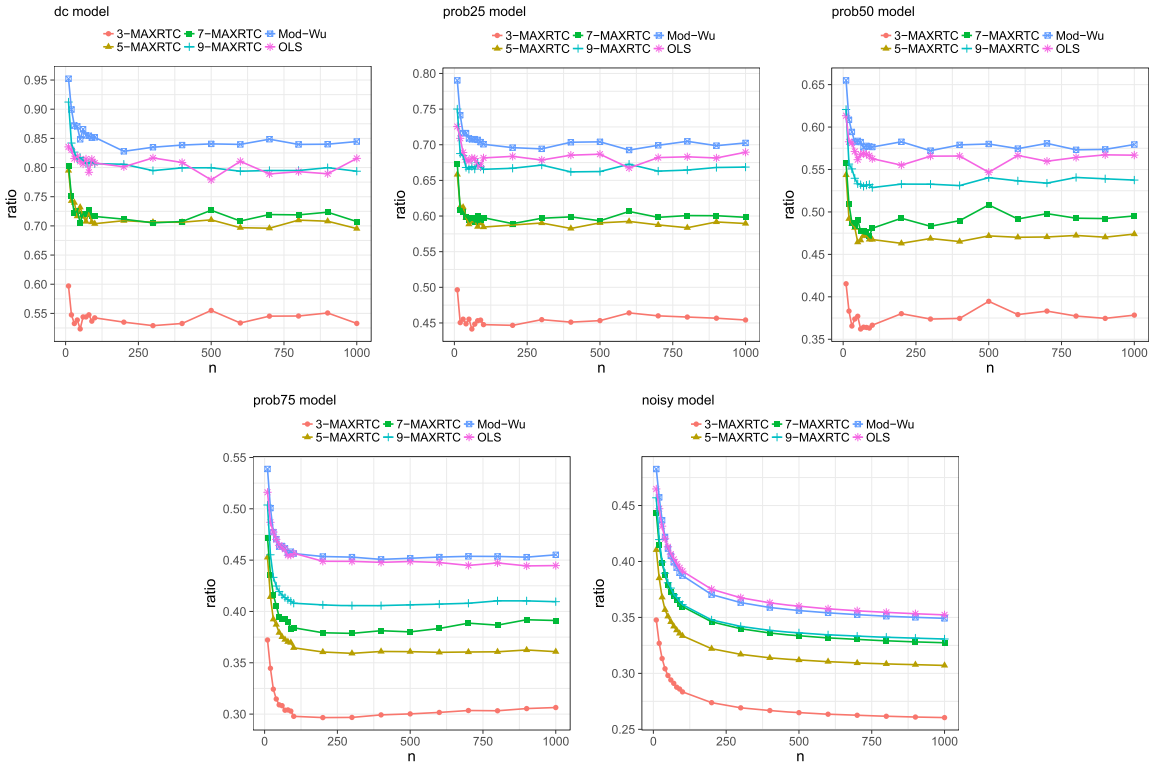


Fig. 4. Performance of $\{3, 5, 7, 9\}$-MAXRTC compared to `OLS` and `Mod-Wu` in the `dc`, `prob25`, `prob50`, `prob75`, and noisy models. Every data point corresponds to the mean of 100 runs. Observe that the performance of 9-MAXRTC is close to that of `OLS` and `Mod-Wu`, even though 9-MAXRTC uses only nine internal nodes while `OLS` uses up to $n-1$ internal nodes and `Mod-Wu` exactly $n-1$. In the noisy model, the performance of q-MAXRTC approaches the theoretical guarantee from Theorem 3.3 (0.25 for $q=3$, 0.296296… for $q=5$, 0.3125 for $q=7$, and 0.32 for $q=9$).

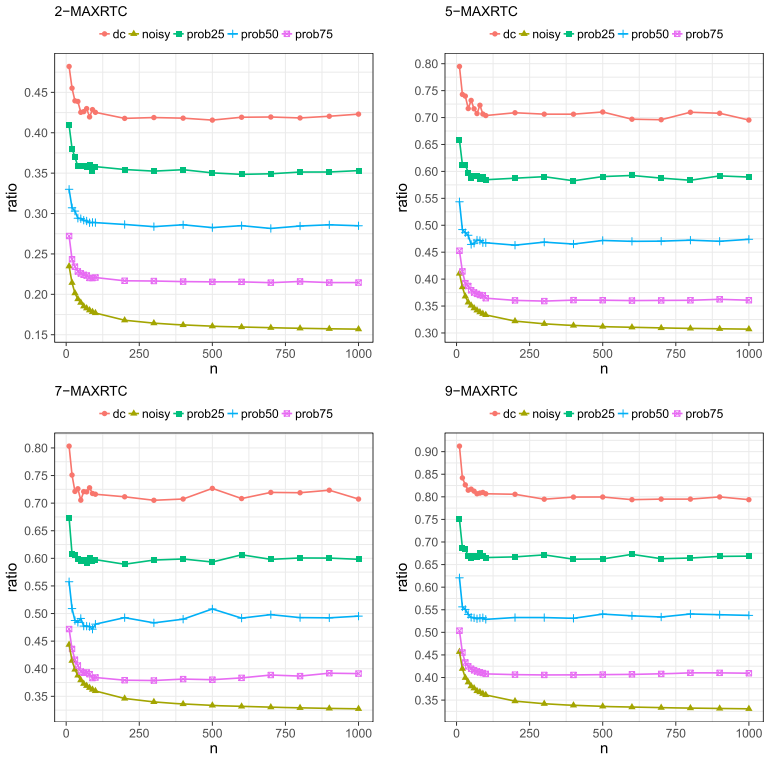


Fig. 5. Performance of {2,5,7,9}-MAXRTC on the different simulated models. Every data point corresponds to the mean of 100 runs.

Table 2
Accuracy of q-MAXRTC on real datasets. Every entry corresponds to the best accuracy over 100 runs. The size of each leaf label set is written inside the parenthesis.

q	poS1(761)	poS2(761)	poS4(841)	nmS4(1869)	nmS2(3082)	Average
2	0.27	0.36	0.43	0.41	0.29	0.35
3	0.67	0.54	0.48	0.41	0.46	0.51
5	0.77	0.81	0.67	0.66	0.72	0.73
7	0.82	0.75	0.76	0.62	0.73	0.74
9	0.86	0.71	0.87	0.80	0.79	0.81
11	0.91	0.89	0.87	0.79	0.87	0.87

Table 3
The execution time (in seconds) to generate each tree produced by q-MAXRTC on the real datasets. Every entry corresponds to the mean of 100 runs. The experiments were performed on a machine with 8GB RAM, Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz, and having Ubuntu 16.04.2 LTS as an operating system.

q	poS1(761)	poS2(761)	poS4(841)	nmS4(1869)	nmS2(3082)
2	0.06	0.06	0.07	0.40	1.16
3	0.32	0.32	0.39	1.96	5.38
5	0.47	0.47	0.58	2.92	7.85
7	0.63	0.62	0.76	3.83	10.53
9	0.78	0.79	0.96	4.85	13.15
11	0.94	0.94	1.14	5.71	15.56

children of nmS4's root to make nmS4 binary as well. The total number of leaves in nmS2, nmS4, poS1, poS2, and poS4 is 1869, 3082, 761, 761, and 841. Since the trees are binary, the total number of internal nodes is 1868, 3081, 760, 760, and 840.

For a tree $T \in \{\texttt{nmS2}, \texttt{nmS4}, \texttt{poS1}, \texttt{poS2}, \texttt{poS4}\}$ with n leaf labels, let T_q be the tree produced by the new algorithm from Theorem 3.3 applied to an appropriate set of triplets, to be defined below. Let $D(T, T_q)$ be the rooted triplet distance between T and T_q (for a definition see Section 5 below). The *accuracy* of q-MAXRTC in the experiments of this dataset is then defined by the ratio $S(T, T_q)/\binom{n}{3}$, where $S(T, T_q) = \binom{n}{3} - D(T, T_q)$. To compute this ratio efficiently, we used the rooted triplet distance implementation in [8]. We measured the accuracy of q-MAXRTC for $q \in \{2, 3, 5, 7, 9, 11\}$. Every experiment consisted of 100 runs, and in each run n^2 triplets were picked at random from the corresponding tree to define the set $\mathcal{R}$. We made sure that each leaf from a given tree appeared in $\mathcal{R}$ so that the size of the leaf label set was as big as the leaf label set of the tree.

Table 2 shows the best ratios achieved, and the corresponding trees in Newick format can be found at https://github.com/kmampent/qMAXRTC/tree/master/trees. As can be seen from the results, a larger number of internal nodes tends to improve the accuracy. Importantly, with only nine nodes we can induce between 71% and 86% of the triplets in each case, and with eleven nodes between 79% and 91%. When $q > 11$, we did not observe a significant improvement in accuracy. Fig. 6 shows the performance of q-MAXRTC on the real datasets, with all points plotted in the right graph. Because of the large sample size, i.e., n^2 triplets were picked at random to define $\mathcal{R}$, the quality of the trees does not differ significantly between the different runs. Finally, Table 3 lists the average number of seconds needed for q-MAXRTC to generate each tree from Table 2. The practicality of the algorithm can be seen from the fact that the largest amount of time spent for a tree is 15.56 seconds, and that is for the nmS2 dataset which contains 3082 leaves and 3081 internal nodes.

5. Motivation for q-MAXRTC: a faster algorithm for the rooted triplet distance

Finally, we give an example of the algorithmic advantage of using phylogenetic trees with few internal nodes. More precisely, we develop an algorithm for computing the rooted triplet distance between two phylogenetic trees in $O(qn)$ time, where q is the number of internal nodes in the smaller tree and n is the number of leaf labels. We also provide an implementation of our new algorithm, along with experimental results comparing the practical performance of our algorithm against previous state-of-the-art algorithms.

5.1. Problem definition

The rooted triplet distance between two trees T_1 and T_2 built on the same leaf label set, is the total number of trees with three leaves that appear as embedded subtrees in T_1 but not in T_2. Intuitively, two trees with very similar branching structure will share many embedded subtrees, so the rooted triplet distance between them will be small.

Formally, let T_1 and T_2 be two trees built on the same leaf label set of size n. We need to distinguish between two types of triplets. The first type is the *resolved triplet*, previously defined in Section 1. In addition, since T_1 and T_2 can be

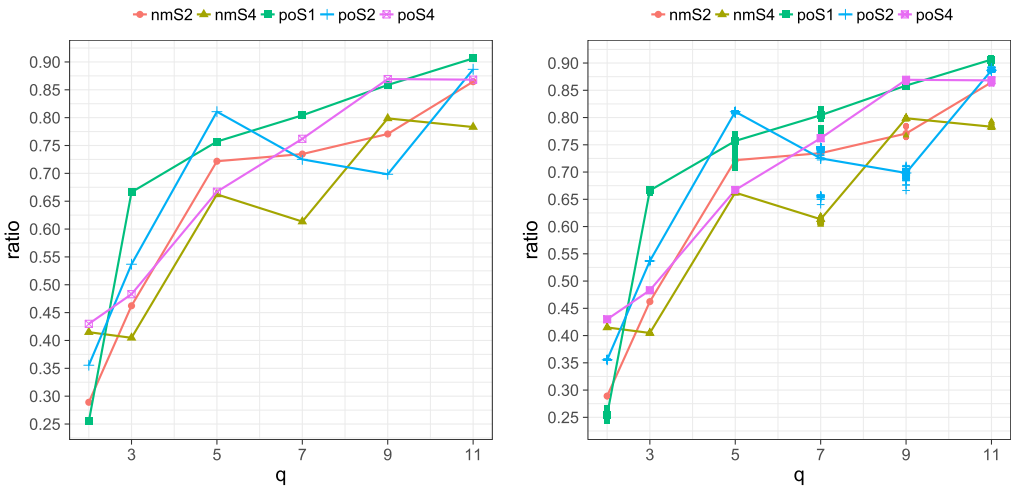


Fig. 6. Graphical representation of the performance of q-MAXRTC on real datasets. Left figure: Every data point corresponds to the mean of 100 runs. Right figure: All 100 data points appear in the figure.

non-binary, we also need to define the *fan triplet*. We call $t = x|y|z$ a *fan triplet*, if t is a tree with the three leaves x, y, and z, and one internal node that is the root of t. The definition of when a resolved triplet is consistent with a tree T follows from Section 1. Analogously, we say that the fan triplet $x|y|z$ is consistent with a tree T, where x, y, and z are leaves in T, if $lca(x, y) = lca(x, z) = lca(y, z)$. In this section only, we use the word *triplet* to refer to both fan and resolved triplets. Moreover, when we refer to a fan triplet $x|y|z$ or a resolved triplet $xy|z$ induced by a tree T, there exists a left to right ordering of x, y, and z in T.

Let $D(T_1, T_2)$ be the rooted triplet distance between T_1 and T_2. Define $S(T_1, T_2)$ to be the total number of triplets that are consistent with both T_1 and T_2, commonly referred to as *shared triplets*. For the rooted triplet distance we then have that $D(T_1, T_2) = \binom{n}{3} - S(T_1, T_2)$.

5.2. The algorithm

It is known how to compute $D(T_1, T_2)$ in $O(n \log n)$ time [7,8]. Below, we show how to compute $D(T_1, T_2)$ in $O(qn)$ time, which is faster than [7,8] when $q = o(\log n)$. The method consists of a preprocessing step and a counting step.

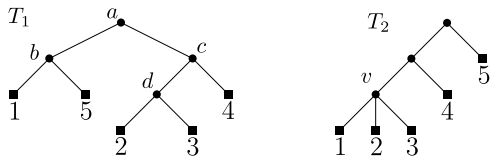
Preprocessing. The leaves in T_2 are relabeled according to their discovery time by a depth first traversal of T_2, in which the children of a node are discovered from left to right. Notice that for a node v in T_2, the labels of the leaves in $T_2(v)$ will correspond to a continuous range of numbers. Afterwards, we transfer the new labels of the leaves in T_2 to the leaves in T_1. For T_1, we define the $q \times n$ table A such that for an internal node u in T_1 we have $A[u][\ell] = 1$ if ℓ is a leaf in $T_1(u)$, and $A[u][\ell] = 0$ otherwise. We construct another table C to answer one dimensional range queries as follows. For $1 \leq i \leq n$ we have $C[u][i] = \sum_{j=1}^{i} A[u][j]$ and $C[u][0] = 0$. The C-table will be used to answer queries asking for the total number of leaves in $T_2(v)$ that are also in $T_1(u)$ in $O(1)$ as follows. Let $[l, \ldots, r]$ be the continuous range of leaf labels in $T_2(v)$. The answer to the query will be exactly $C[u][r] - C[u][l-1]$ (see Fig. 7 for an example).

Counting. We extend the technique introduced in [8]. Let $t = xy|z$ or $t = x|y|z$ be a triplet induced by a tree T, which in our problem can be either T_1 or T_2. We anchor t in the edge $\{v, c\}$, where $v = lca(x, y)$ and c is the child of v such that $T(v)$ contains y. The following lemma shows that every triplet induced by T is anchored in exactly one edge of T.

Lemma 5.1. *Let T be a tree in which every triplet t with the three leaves x, y, and z is anchored in the edge $\{u, c\}$, such that $u = lca(x, y)$ and $T(c)$ contains y. Every triplet induced by T is anchored in exactly one edge of T.*

Proof. For the purpose of obtaining a contradiction, assume that t is anchored in i edges where $i \geq 2$. Let those edges be $\{u_1, v_1\}, \ldots, \{u_i, v_i\}$. By definition, $u_1 = lca(x, y)$. For the same reason, $u_2 = lca(x, y), \ldots, u_i = lca(x, y)$. Because T is a tree, it must hold that $u_1 = u_2 = \ldots = u_i$. Again by definition, $v_1, \ldots, v_i$ must all be the children of u_1 such that every subtree in $S = \{T(v_1), \ldots, T(v_i)\}$ contains leaf y. Because T is a tree, there can only be one such child of u_1, meaning that $|S| = 1$, which leads to a contradiction. □

Suppose that a node v in T_2 has the children $v_1, \ldots, v_j, \ldots, v_i$, where $1 < j \leq i$. To detect all triplets anchored in edge $\{v, v_j\}$ of T_2, we color the leaves of T_2 as follows. Let every leaf in $T_2(v_1), \ldots, T_2(v_{j-1})$ have the color red, every leaf in $T_2(v_j)$ have the color blue, every leaf in $T_2(v_{j+1}), \ldots, T_2(v_i)$ have the color green, and all other leaves in T_2 have the color white. The red, blue, and green colors will be used to detect fan triplets and the red, blue, and white colors, resolved triplets. See Fig. 8 for an illustration. By the relabeling scheme of the leaves, we have that the red, blue, and green colors



	1	2	3	4	5
a	1	1	1	1	1
b	1	0	0	0	1
c	0	1	1	1	0
d	0	1	1	0	0

A

	0	1	2	3	4	5
a	0	1	2	3	4	5
b	0	1	1	1	1	2
c	0	0	1	2	3	3
d	0	0	1	2	2	2

C

Fig. 7. Changing the leaf labels of the trees from Fig. 1 according to the preprocessing step of the rooted triplet distance algorithm in Section 5. The leaves in $T_2(v)$ are defined by the range of leaf labels $[1, 3]$. The number of leaves appearing in both $T_2(v)$ and $T_1(c)$ is $C[c][3] - C[c][0] = 2$.

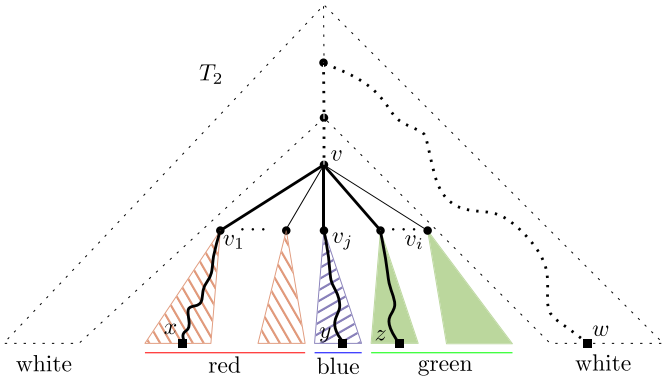


Fig. 8. Using colors to detect triplets in T_2. Three leaves x, y, and z that have the color red, blue, and green respectively, define the fan triplet $x|y|z$. Three leaves x, y, and z that have the color red, blue, and white respectively, define the resolved triplet $xy|z$.

correspond to exactly one continuous range of leaf labels each. Let those ranges be $R = [a_{\text{red}}, \ldots, a'_{\text{red}}]$, $B = [a_{\text{blue}}, \ldots, a'_{\text{blue}}]$, and $G = [a_{\text{green}}, \ldots, a'_{\text{green}}]$, for the colors red, blue, and green respectively. We have $a_{\text{blue}} = a'_{\text{red}} + 1$ and if G is non-empty, $a_{\text{green}} = a'_{\text{blue}} + 1$. Finally, note that a leaf has the color white if and only if it does not have any other color. We now describe how to compute the total number of triplets anchored in some edge $\{v, v'\}$ in T_2, where v is the parent of v', that are also consistent with T_1, denoted $S^{\{v,v'\}}(T_1, T_2)$. Let $S_r^{\{v,v'\}}(T_1, T_2)$ be the number of shared resolved triplets anchored in $\{v, v'\}$ and similarly let $S_f^{\{v,v'\}}(T_1, T_2)$ be the number of shared fan triplets. Clearly, $S^{\{v,v'\}}(T_1, T_2) = S_r^{\{v,v'\}}(T_1, T_2) + S_f^{\{v,v'\}}(T_1, T_2)$. The following lemma shows that $S^{\{v,v'\}}(T_1, T_2)$ can be computed efficiently.

Lemma 5.2. *Given the ranges R, B, and G that define a coloring of the leaves in T_2 according to an edge $\{v, v'\}$ of T_2, there exists an $O(q)$-time algorithm for computing $S^{\{v,v'\}}(T_1, T_2)$.*

Proof. Since both T_1 and T_2 are built on the same leaf label set, a coloring of the leaves of T_2 defines a coloring of the leaves of T_1. Suppose that a node u in T_1 has the m children $u_1, \ldots, u_m$, where $m \geq 2$. Some children could be leaves and others, internal nodes. Let I denote the set containing the children that are internal nodes and L the children that are leaves. Let $T(I) = \bigcup_{u \in I} \{T(u)\}$. Define the following counters:

1. u_{white}: total number of leaves with the white color in T_1 but not in $T_1(u)$.
2. u_i, for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in $T_1(u)$.
3. u_{iI}, for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in $T(I)$.
4. u_{iL}, for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in L.
5. $u_{i,j}$, for $(i, j) \in \{(\textit{red}, \textit{blue}), (\textit{red}, \textit{green}), (\textit{blue}, \textit{green})\}$: total number of pairs of leaves in $T(I)$, such that one has color i, the other has color j and both come from different subtrees attached to u.
6. $u_{\text{red, blue, green}}$: total number of leaf triples in $T(I)$, such that one leaf has the color red, another the color blue, another the color green and they all come from different subtrees attached to u.

Algorithm 4 O(q)-time algorithm for computing the counters of every internal node u in T_1 (see Lemma 5.2).

1: **procedure** CountersHelper(u, C, R, B, G)
2: Let $R = [a_{\text{red}}, \ldots, a'_{\text{red}}]$, $B = [a_{\text{blue}}, \ldots, a'_{\text{blue}}]$, $G = [a_{\text{green}}, \ldots, a'_{\text{green}}]$
3: Let u_l denote the total number of leaves in $T_1(u)$
4: totalWhite $= n - (a'_{\text{red}} - a_{\text{red}} + 1) - (a'_{\text{blue}} - a_{\text{blue}} + 1) - (a'_{\text{green}} - a_{\text{green}} + 1)$
5: **if** u has no children that are internal nodes **then**
6: **for** every color $i \in$ {red, blue, green} **do**
7: $u_i = C[u][a'_i] - C[u][a_i - 1]$
8: $u_{iI} = 0$
9: $u_{iL} = u_i$
10: $u_{\text{white}} =$ totalWhite $- u_l + u_{\text{red}} + u_{\text{blue}} + u_{\text{green}}$
11: $u_{\text{red,blue}} = u_{\text{red,green}} = u_{\text{blue,green}} = 0$
12: $u_{\text{red,blue,green}} = 0$
13: **return**
14: Let I denote the set of children of u that are internal nodes in T_1
15: **for** every node w in I **do**
16: CountersHelper(w, C, R, B, G)
17: pick some node w from I
18: **for** every color $i \in$ {red, blue, green} **do**
19: $u_i = C[u][a'_i] - C[u][a_i - 1]$
20: $u_{iI} = w_i$
21: $u_{iL} = u_i$
22: **for** every node w' in I **do**
23: $u_{iL} = u_{iL} - (C[w'][a'_i] - C[w'][a_i - 1])$
24: $u_{\text{white}} =$ totalWhite $- u_l + u_{\text{red}} + u_{\text{blue}} + u_{\text{green}}$
25: $u_{\text{red,blue}} = u_{\text{red,green}} = u_{\text{blue,green}} = 0$
26: $u_{\text{red,blue,green}} = 0$
27: remove w from I
28: **while** I is not empty **do**
29: pick some node w from I
30: $u_{\text{red,blue,green}} = u_{\text{red,blue,green}} + u_{\text{red,blue}} \cdot w_{\text{green}} + u_{\text{red,green}} \cdot w_{\text{blue}} + u_{\text{blue,green}} \cdot w_{\text{red}}$
31: $u_{\text{red,blue}} = u_{\text{red,blue}} + u_{\text{red}I} \cdot w_{\text{blue}} + u_{\text{blue}I} \cdot w_{\text{red}}$
32: $u_{\text{red,green}} = u_{\text{red,green}} + u_{\text{red}I} \cdot w_{\text{green}} + u_{\text{green}I} \cdot w_{\text{red}}$
33: $u_{\text{blue,green}} = u_{\text{blue,green}} + u_{\text{blue}I} \cdot w_{\text{green}} + u_{\text{green}I} \cdot w_{\text{blue}}$
34: $u_{\text{red}I} = u_{\text{red}I} + w_{\text{red}}$
35: $u_{\text{blue}I} = u_{\text{blue}I} + w_{\text{blue}}$
36: $u_{\text{green}I} = u_{\text{green}I} + w_{\text{green}}$
37: remove w from I
38: **procedure** Counters(T_1, C, R, B, G)
39: let r be the root of T_1
40: CountersHelper(r, C, R, B, G)

Algorithm 4 shows how to compute these counters for every internal node of T_1. A depth first traversal is applied on T_1 while making sure that we only visit internal nodes. For every node u in I we apply a dynamic programming procedure (lines 17-37) to compute the counters. Since in every recursive call we spend O($|I|$) time, the total time of the algorithm is O(q).

After computing all counters in T_1 by applying Algorithm 4, Algorithm 5 shows how to compute $S_f^{\{v,v'\}}(T_1, T_2)$ and $S_r^{\{v,v'\}}(T_1, T_2)$ in O(q) time as well.

Algorithm 5 Computing $S_f^{\{v,v'\}}(T_1, T_2)$ and $S_r^{\{v,v'\}}(T_1, T_2)$ in O(q) time.

1: **procedure** $S_f^{\{v,v'\}}(T_1, T_2)$
2: fans $= 0$
3: **for** every internal node u in T_1 **do**
4: fans $=$ fans $+ u_{\text{red,blue,green}}$
5: fans $=$ fans $+ u_{\text{red,blue}} \cdot u_{\text{green}L} + u_{\text{red,green}} \cdot u_{\text{blue}L} + u_{\text{blue,green}} \cdot u_{\text{red}L}$
6: fans $=$ fans $+ u_{\text{red}I} \cdot u_{\text{blue}L} \cdot u_{\text{green}L} + u_{\text{blue}I} \cdot u_{\text{red}L} \cdot u_{\text{green}L} + u_{\text{green}I} \cdot u_{\text{red}L} \cdot u_{\text{blue}L}$
7: fans $=$ fans $+ u_{\text{red}L} \cdot u_{\text{blue}L} \cdot u_{\text{green}L}$
8: **return** fans
9: **procedure** $S_r^{\{v,v'\}}(T_1, T_2)$
10: resolved $= 0$
11: **for** every internal node u in T_1 **do**
12: resolved $=$ resolved $+ u_{\text{red,blue}} \cdot u_{\text{white}}$
13: resolved $=$ resolved $+ u_{\text{red}I} \cdot u_{\text{blue}L} \cdot u_{\text{white}} + u_{\text{blue}I} \cdot u_{\text{red}L} \cdot u_{\text{white}}$
14: resolved $=$ resolved $+ u_{\text{red}L} \cdot u_{\text{blue}L} \cdot u_{\text{white}}$
15: **return** resolved

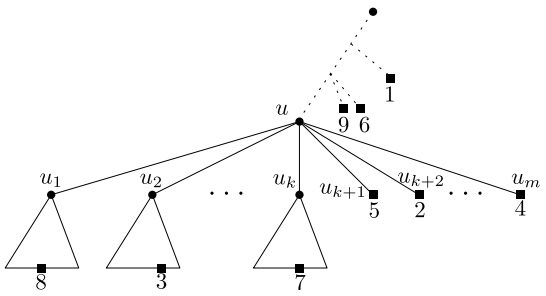


Fig. 9. Rooted triplet distance computation: an internal node u in T_1, having m children k of which are internal nodes. In Algorithm 5, $I = \{u_1, \ldots, u_k\}$ and $L = \{u_{k+1}, \ldots, u_m\}$.

Algorithm 5 counts shared triplets by considering for every internal node u in T_1, all possible cases for the location of the leaves of a shared triplet anchored in any edge $\{u, u'\}$ in T_1, where u is the parent of u'. More precisely, for the leaves of a fan triplet anchored in any edge $\{u, u'\}$ in T_1, we have the following cases:

1. all three leaves come from $T(I)$ (line 4, e.g., 8|3|7 in Fig. 9).
2. two leaves come from $T(I)$ and one from L (line 5, e.g., 7|3|2 in Fig. 9).
3. one leaf comes from $T(I)$ and two from L (line 6, e.g., 3|5|4 in Fig. 9).
4. all three leaves come from L (line 7, e.g., 4|2|5 in Fig. 9).

Similarly, for the leaves of a resolved triplet we have the following cases:

1. two leaves come from $T(I)$ and one not from $T_1(u)$ (line 11, e.g., 38|6 in Fig. 9).
2. one leaf comes from $T(I)$, one from L, and one not from $T_1(u)$ (line 12, e.g., 35|1 in Fig. 9).
3. two leaves come from L and one not from $T_1(u)$ (line 13, e.g., 52|9 in Fig. 9).

Since $S^{\{v,v'\}}(T_1, T_2) = S_r^{\{v,v'\}}(T_1, T_2) + S_f^{\{v,v'\}}(T_1, T_2)$, the lemma statement follows. □

Finally, Algorithm 6 computes $D(T_1, T_2)$. From the preprocessing step, line 2 requires $O(qn)$ time. Line 3 is performed by a depth first traversal of T_1, thus requiring $O(n)$ time. From Lemma 5.2, lines 7-9 require $O(q)$ time. Since we also have that $\sum_{v \in T_2} deg(v) = O(n)$, the total time required to compute $D(T_1, T_2)$ is $O(qn)$. The correctness is ensured by Lemma 5.1, thus we obtain the following theorem.

Algorithm 6 $O(qn)$-time algorithm for computing $D(T_1, T_2)$.

```
1:  procedure D(T1, T2)
2:      Compute the q × n table C.
3:      For every node u in T1, compute u_l (= the number of leaves in T1(u)).
4:      shared = 0
5:      for every internal node v in T2 do
6:          for every child v' of v do
7:              Let R, B, and G be the color ranges defined by edge {v, v'}
8:              COUNTERS(T1, C, R, B, G)                          ▷ From Lemma 5.2
9:              shared = shared + S_f^{v,v'}(T1, T2)+S_r^{v,v'}(T1, T2)
10:     return (n choose 3) − shared
```

Theorem 5.1. *The rooted triplet distance between two rooted phylogenetic trees T_1 and T_2 built on the same leaf label set of size n, can be computed in $O(qn)$ time, where q is the total number of internal nodes in T_1.*

5.3. Implementation and experiments

We implemented the $O(qn)$-time triplet distance algorithm in the C++ programming language. The source code is available at https://github.com/kmampent/qtd. The experiments were performed on a machine with 8GB RAM, Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz, and having Ubuntu 16.04.2 LTS as an operating system. For the space usage we considered the *Maximum resident set size* parameter returned by /usr/bin/time -v. We ran the experiments using the following two different models for generating input trees:

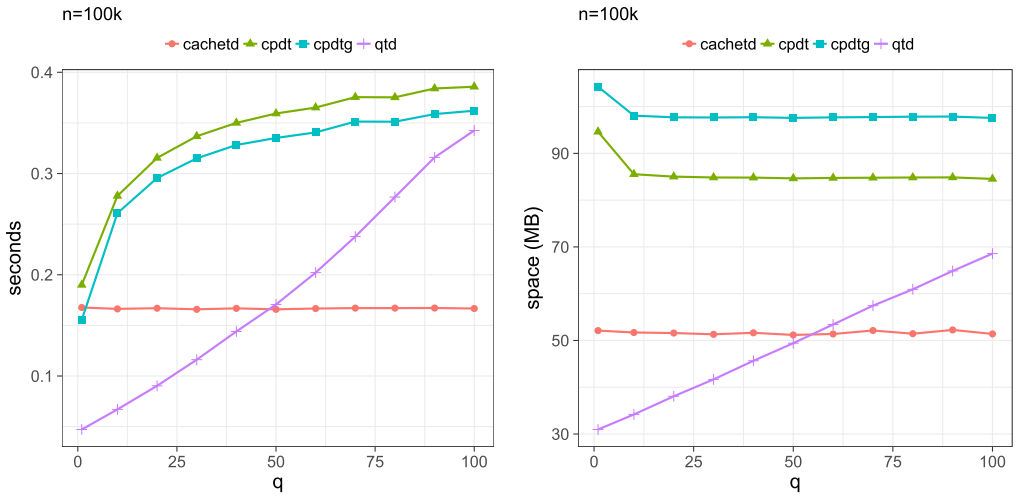


Fig. 10. Model A: Running time and space usage of the different triplet distance algorithms when $n = 10^5$.

- *Model A*: Generate a binary tree with n leaves for T_2 following the uniform model [27,29]. For T_1, first create a tree with q internal nodes as follows: start with a tree containing one internal node, then as long as T_1 does not have q internal nodes, pick an internal node u uniformly at random and insert a new internal node as a child of u. Next, to assign the leaves, first assign one leaf to every internal node that has only one child and then assign any remaining leaves to internal nodes selected uniformly at random.
- *Model B*: Generate a binary tree with n leaves for T_2 following the uniform model [27,29]. For T_1, start by generating a binary tree with n leaves following the uniform model. Let S be a set containing the root and $q-1$ internal nodes picked uniformly at random. Contract every internal node u that is not in S by letting the children of u become the children of u's parent and then deleting u.

We compared our algorithm against the $O(n \log n)$-time algorithm in [8] and the $O(n \log^3 n)$-time algorithm in [21]. We call our algorithm `qtd` and the algorithm in [8] `cachetd`. In [21] two different implementations are provided, one that uses unordered_map,[1] which we refer to as `cpdt`, and one that uses sparsehash,[2] which we refer to as `cpdtg`.

Every data point in the figures corresponds to the mean of 50 different runs, each run on a different pair of input trees. In Fig. 10 we have the execution time in seconds as well as the space usage of the algorithms in model A and when $n = 10^5$. In Fig. 12 we have the same experiments except now $n = 10^6$. Figs. 11 and 13 have the previous experiments but in model B. The results indicate that our implementation uses less space and is faster than the previous algorithms when $q \leq 50$.

6. Concluding remarks

In this article, we introduced the problem of building a phylogenetic tree with q internal nodes that induces the largest number of triplets from an input triplet set $\mathcal{R}$, denoted q-MAXRTC. We showed that q-MAXRTC is NP-hard for every fixed $q \geq 2$. For 2-MAXRTC, no polynomial-time approximation algorithm with a relative (resp. absolute) approximation ratio better than $\frac{16}{17} + \epsilon$ (resp. $\frac{4}{27} + \epsilon$) can exist. When $q \geq 3$, the inapproximability bound becomes $1 - 1/(34q) + \epsilon$. We reduced 2-MAXRTC to MAX 3-AND and obtained several polynomial-time approximation algorithms that, however, could not scale with q. We then proposed a simple polynomial-time $\frac{4}{27}$-approximation algorithm that could be extended to scale with any $q \geq 3$. Finally, for two trees with one having q internal nodes and both being built on the same leaf label set of size n, we presented an $O(qn)$-time algorithm for the rooted triplet distance computation. We have also implemented the $O(qn)$-time algorithm described in Section 5. Our experiments indicate that our prototype implementation uses less space and is faster than the state-of-the-art, optimized implementation of the $O(n \log n)$-time algorithm from [8] for large inputs, e.g., when $n = 10^6$ and $q \leq 50$.

Some open problems are:

- To derive the optimal polynomial-time approximation ratio for q-MAXRTC for any fixed $q \geq 3$ is an open problem, as well as to develop an efficient algorithm achieving that ratio. To design efficient approximation algorithms for the *weighted* version of q-MAXRTC is also an open problem, where every triplet in $\mathcal{R}$ has a weight and the objective is to

[1] http://en.cppreference.com/w/cpp/container/unordered_map.
[2] https://github.com/sparsehash/sparsehash.

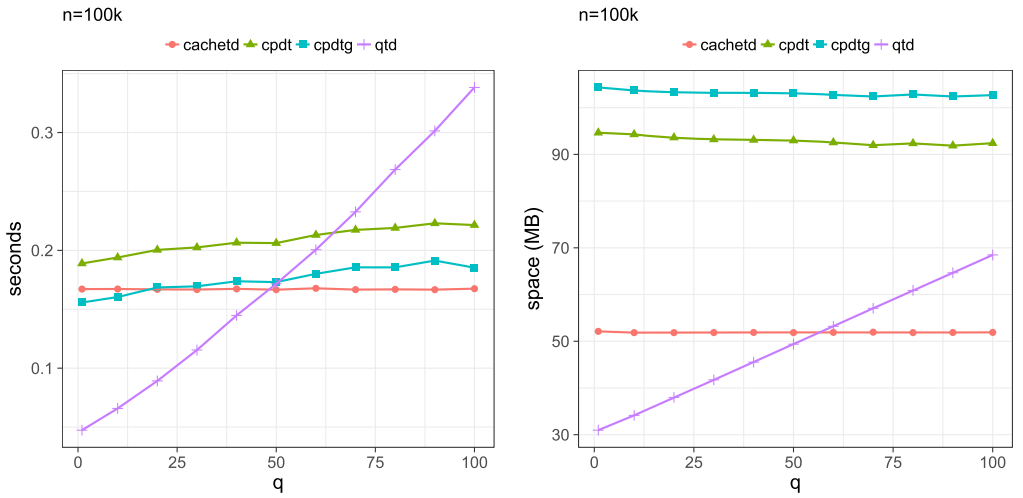


Fig. 11. Model B: Running time and space usage of the different triplet distance algorithms when $n = 10^5$.

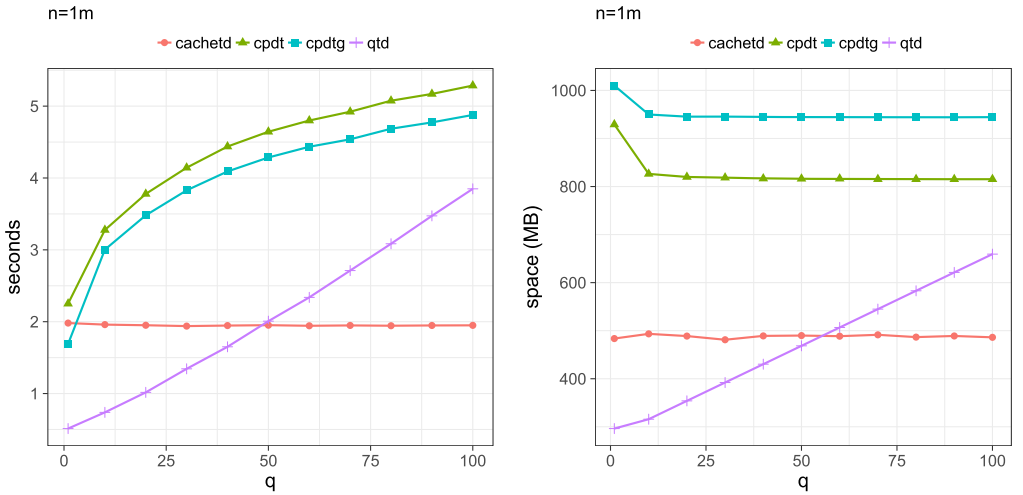


Fig. 12. Model A: Running time and space usage of the different triplet distance algorithms when $n = 10^6$.

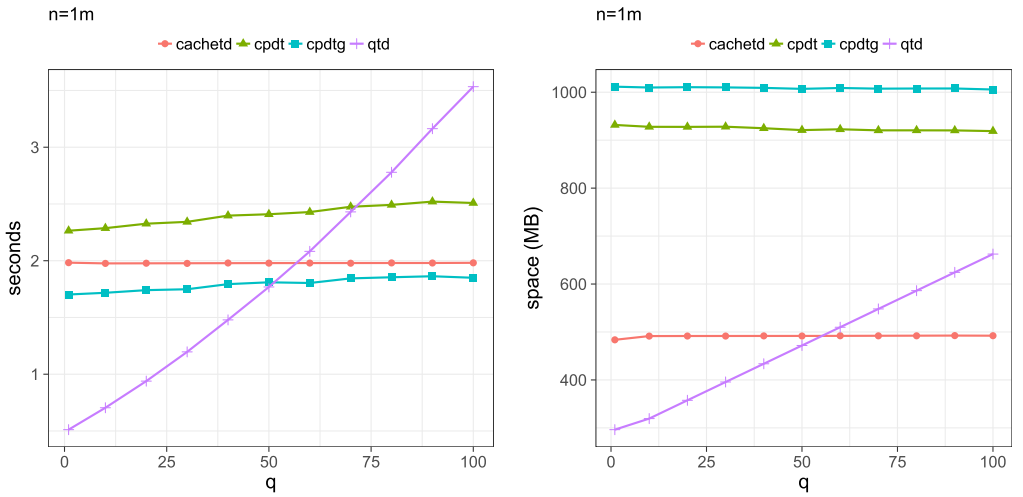


Fig. 13. Model B: Running time and space usage of the different triplet distance algorithms when $n = 10^6$.

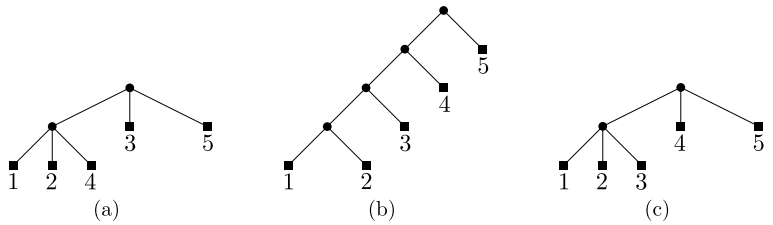


Fig. 14. Let $\mathcal{R} = \{12|3, 13|4, 24|5\}$. (a) The optimal tree for 2-MAXRTC induces two triplets from $\mathcal{R}$. (b) The tree returned by the BUILD algorithm [2] when applied to $\mathcal{R}$. (c) The best tree obtainable by contracting all internal edges except one in the tree from (b) induces only one triplet from $\mathcal{R}$, so this method is not optimal for 2-MAXRTC.

build a tree that maximizes the total weight of its induced triplets that belong to $\mathcal{R}$. This would be useful in situations where some triplets in $\mathcal{R}$ are known to be more reliable than others or where certain triplets should be given a higher priority.

- The computational complexity of q-MAXRTC restricted to inputs where all triplets in $\mathcal{R}$ are consistent with some tree T having an unbounded number of internal nodes is still unknown. A simple idea might be to first run the BUILD algorithm [2] to obtain such a T and then try every bipartition of L induced by an edge of T, but as demonstrated in Fig. 14, this approach would fail to produce optimal solutions even for 2-MAXRTC. The issue in this particular example seems to be that while running BUILD, nothing indicates that it would be better to discard leaf 3 from the connected component $\{1, 2, 3, 4\}$ before leaf 4.
- Another related problem that has not yet been studied is the following: Given a set of triplets $\mathcal{R}$ on a leaf label set of size n and a parameter ℓ, output a tree T with ℓ leaves such that $|rt(T) \cap \mathcal{R}|$ is maximized. Just like q-MAXRTC is a combination of MINRS and MAXRTC, this new problem is a combination of the *maximum agreement supertree problem* from [5,20] and MAXRTC.
- For the rooted triplet distance computation, a major open problem [7,8] is whether it can be computed in $O(n)$ time. When $q = O(1)$, our proposed algorithm runs in $O(n)$ time. Is it possible to refine it to run in $O(q_1 q_2 + n)$ time, where q_1 (resp. q_2) is the number of internal nodes in T_1 (resp. T_2)?
- A general problem inspired by the work in this article is: Given a tree T and an integer q, what does the tree T' having q internal nodes that represents T in the best way possible look like? In other words, how should a tree T be simplified to a smaller tree T' with at most q internal nodes while keeping as much important information as possible? One can consider different alternatives when defining "in the best way possible" and "important information", not necessarily based on triplets.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

JJ was partially funded by RGC/GRF project 15217019 and KAKENHI grant 22H03550, and STP by PolyU Fund 1-ZE8L. KM acknowledges the support by the Danish National Research Foundation, grant DNRF84, via the Center for Massive Data Algorithmics (MADALGO).

References

[1] E.N. Adams III, Consensus techniques and the comparison of taxonomic trees, Syst. Zool. 21 (4) (1972) 390–397.
[2] A.V. Aho, Y. Sagiv, T.G. Szymanski, J.D. Ullman, Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions, SIAM J. Comput. 10 (3) (1981) 405–421.
[3] P. Alimonti, New local search approximation techniques for maximum generalized satisfiability problems, Inf. Process. Lett. 57 (3) (1996) 151–158.
[4] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: LATIN 2000: Theoretical Informatics, Springer, Berlin Heidelberg, 2000, pp. 88–94.
[5] V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, J. Discret. Algorithms 5 (3) (2007) 564–591.
[6] O.R.P. Bininda-Emonds, The evolution of supertrees, Trends Ecol. Evol. 19 (6) (2004) 315–322.
[7] G.S. Brodal, R. Fagerberg, C.N.S. Pedersen, T. Mailund, A. Sand, Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree, in: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2013, pp. 1814–1832.
[8] G.S. Brodal, K. Mampentzidis, Cache oblivious algorithms for computing the triplet distance between trees, ACM J. Exp. Algorithmics 26 (2021) 1.2.
[9] D. Bryant, Building Trees, Hunting for Trees, and Comparing Trees - Theory and Methods in Phylogenetic Analysis, PhD thesis, University of Canterbury, Christchurch, NZ, 1997.
[10] D. Bryant, A classification of consensus methods for phylogenetics, in: M.F. Janowitz, F.-J. Lapointe, F.R. McMorris, B. Mirkin, F.S. Roberts (Eds.), Bioconsensus, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, American Mathematical Society, 2003, pp. 163–184.
[11] J. Byrka, P. Gawrychowski, K.T. Huber, S. Kelk, Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks, J. Discret. Algorithms 8 (1) (2010) 65–75.

[12] J. Byrka, S. Guillemot, J. Jansson, New results on optimizing rooted triplets consistency, Discrete Appl. Math. 158 (11) (2010) 1136–1147.
[13] B. Chor, M. Hendy, D. Penny, Analytic solutions for three taxon ML trees with variable rates across sites, Discrete Appl. Math. 155 (6) (2007) 750–758.
[14] J. Felsenstein, Inferring Phylogenies, Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.
[15] L. Gąsieniec, J. Jansson, A. Lingas, A. Östlin, On the complexity of constructing evolutionary trees, J. Comb. Optim. 3 (2) (1999) 183–197.
[16] J. Håstad, Some optimal inapproximability results, J. ACM 48 (4) (2001) 798–859.
[17] M.R. Henzinger, V. King, T. Warnow, Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology, Algorithmica 24 (1) (1999) 1–13.
[18] L.A. Hug, B.J. Baker, K. Anantharaman, C.T. Brown, A.J. Probst, C.J. Castelle, C.N. Butterfield, A.W. Hernsdorf, Y. Amano, K. Ise, Y. Suzuki, N. Dudek, D.A. Relman, K.M. Finstad, R. Amundson, B.C. Thomas, J.F. Banfield, A new view of the tree of life, Nat. Microbiol. 1 (2016) 16048.
[19] J. Jansson, R.S. Lemence, A. Lingas, The complexity of inferring a minimally resolved phylogenetic supertree, SIAM J. Comput. 41 (1) (2012) 272–291.
[20] J. Jansson, J.H.-K. Ng, K. Sadakane, W.-K. Sung, Rooted maximum agreement supertrees, Algorithmica 43 (4) (2005) 293–307.
[21] J. Jansson, R. Rajaby, A more practical algorithm for the rooted triplet distance, J. Comput. Biol. 24 (2) (2017) 106–126.
[22] J. Jansson, R. Rajaby, W.-K. Sung, Minimal phylogenetic supertrees and local consensus trees, AIMS Med. Sci. 5 (medsci-05–02–181) (2018) 181.
[23] V. Kann, S. Khanna, J. Lagergren, A. Panconesi, On the hardness of approximating max k-cut and its dual, Chic. J. Theor. Comput. Sci. (1997).
[24] A.M. Kerr, Molecular and morphological supertree of stony corals (Anthozoa: Scleractinia) using matrix representation parsimony, Biol. Rev. 80 (4) (2005) 543–558.
[25] J.M. Lang, A.E. Darling, J.A. Eisen, Phylogeny of bacterial and archaeal genomes using conserved genes: supertrees and supermatrices, PLoS ONE 8 (4) (2013) e62510.
[26] K.J. Locey, J.T. Lennon, Scaling laws predict global microbial diversity, Proc. Natl. Acad. Sci. 113 (21) (2016) 5970–5975.
[27] A. McKenzie, M. Steel, Distributions of cherries for two models of trees, Math. Biosci. 164 (1) (2000) 81–92.
[28] L. Nakhleh, T. Warnow, D. Ringe, S.N. Evans, A comparison of phylogenetic reconstruction methods on an indo-European dataset, Trans. Philol. Soc. 103 (2) (2005) 171–192.
[29] C. Semple, M. Steel, Phylogenetics, Oxford Lecture Series in Mathematics and Its Applications, vol. 24, Oxford University Press, 2003.
[30] L. Trevisan, Parallel approximation algorithms by positive linear programming, Algorithmica 21 (1) (1998) 72–88.
[31] D.P. Williamson, D.B. Shmoys, The Design of Approximation Algorithms, 1st edition, Cambridge University Press, 2011, pp. 108–109.
[32] B.Y. Wu, Constructing the maximum consensus tree from rooted triples, J. Comb. Optim. 8 (1) (2004) 29–39.
[33] U. Zwick, Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint, in: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '98, Society for Industrial and Applied Mathematics, 1998, pp. 201–210.