Building a Small and Informative Phylogenetic Supertree

Jesper Jansson
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
jesper.jansson@polyu.edu.hk

Konstantinos Mampentzidis
Department of Computer Science, Aarhus University, Aarhus, Denmark
kmampent@cs.au.dk

Sandhya T. P.
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
tp.sandhya@gmail.com

Abstract

We combine two fundamental, previously studied optimization problems related to the construction of phylogenetic trees called *maximum rooted triplets consistency* (MAXRTC) and *minimally resolved supertree* (MINRS) into a new problem, which we call *q-maximum rooted triplets consistency* (q-MAXRTC). The input to our new problem is a set $\mathcal{R}$ of resolved triplets (rooted, binary phylogenetic trees with three leaves each) and the objective is to find a phylogenetic tree with exactly q internal nodes that contains the largest possible number of triplets from $\mathcal{R}$. We first prove that q-MAXRTC is NP-hard even to approximate within a constant ratio for every fixed $q \geq 2$, and then develop various polynomial-time approximation algorithms for different values of q. Next, we show experimentally that representing a phylogenetic 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. Finally, to demonstrate the algorithmic advantage of using trees with few internal nodes, we propose a new algorithm for computing the *rooted triplet distance* between two phylogenetic trees over a leaf label set of size n that runs in $O(qn)$ time, where q is the number of internal nodes in the smaller tree, and is therefore faster than the currently best algorithms for the problem (with $O(n \log n)$ time complexity [SODA 2013, ESA 2017]) whenever $q = o(\log n)$.

2012 ACM Subject Classification Mathematics of computing → Trees

Keywords and phrases phylogenetic tree, supertree, rooted triplet, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.1

Funding Konstantinos Mampentzidis was funded by the Danish National Research Foundation, grant DNRF84, via the Center for Massive Data Algorithmics (MADALGO).

1 Introduction

Background. Phylogenetic trees are used in biology to represent evolutionary relationships. The leaves in such a tree correspond to species that exist today and internal nodes to ancestor species that existed in the past. 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 [3] 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. Depending on the type of data that is available and the type of trees that we want to construct, we obtain several variants of the same problem.

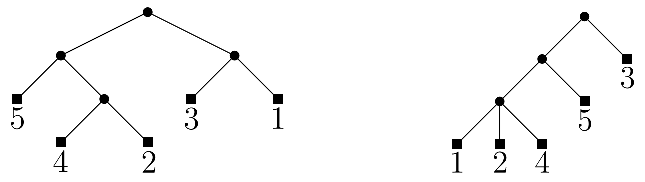

Figure 1 Let $L = \{1,2,3,4,5\}$ and $\mathcal{R} = \{45|3, 25|3, 13|5, 24|5, 23|1\}$. In this example no tree T such that $|\mathcal{R} \cap rt(T)| = 5$ exists. Left figure: optimal solution for MAXRTC with value 4. Right figure: optimal solution for 3-MAXRTC with value 3.

Problem Definition. A *rooted phylogenetic tree* is a rooted unordered tree in which every leaf has a distinct label and every internal node has at least two children. In this article, for simplicity we use the word "tree" to refer to a "rooted phylogenetic tree". A *resolved triplet* is a binary tree with three leaves. The resolved triplet with leaf labels x, y, and z where z is closest to the root is denoted by $xy|z$. From now on when referring to a "triplet" we mean a "resolved triplet". Let T be a tree on a leaf label set L of size n. For a node $u \in T$, $deg(u)$ is the number of children of u and $T(u)$ is the subtree induced by u and all the proper descendants of u. For two nodes u and v in T, $lca(u,v)$ is the lowest common ancestor node of u and v in T. We say that the triplet $xy|z$ is *induced by* T if $lca(x,z) = lca(y,z)$ and $lca(x,y) \neq lca(x,z)$. Let $rt(T)$ be the set of all triplets induced by T. Given a set $\mathcal{R}$ of triplets, we say that $\mathcal{R}$ *is consistent with* T, or equivalently T *is consistent with* $\mathcal{R}$, if $\mathcal{R} \subseteq rt(T)$.

Given a set $\mathcal{R}$ of triplets on a leaf label set L of size n, in the *q-maximum rooted triplets consistency problem*, denoted q-MAXRTC, the goal is to find a tree T with exactly q internal nodes such that $|\mathcal{R} \cap rt(T)|$ is maximized, i.e., the total number of triplets induced by T that are also in $\mathcal{R}$ is as large as possible. An example can be seen in Figure 1.

Let A be an algorithm for any 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.

Previous Work. Aho et al. [1] 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 [6], 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 [12], which is undesirable because unnecessary internal nodes may suggest false groupings of the leaves, also known as *spurious novel clades* [3]. Moreover, scientists typically look for the simplest possible explanation for some given observations and would prefer a tree that makes as few additional statements as possible about evolutionary relationships that are not supported by the input data. 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 $q \leq 3$ [12], where q is the total number of internal nodes 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 [14]. 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 for any q [12].

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 [6] showed that MAXRTC is NP-hard and Gąsieniec et al. [10] proposed a polynomial-time top-down $\frac{1}{3}$-approximation algorithm that always returns a caterpillar tree. Byrka et al. [8] showed that a bottom-up algorithm by Wu [21] can be modified to also obtain a polynomial-time $\frac{1}{3}$-approximation algorithm. In [7], Byrka et al. gave a $\frac{1}{3}$-approximation algorithm by derandomizing a randomized algorithm. In Section 3 below, we refer to the algorithm in [10] as *One-Leaf-Split* (`OLS`) and the algorithm in [8] as *Wu's algorithm* (`WU`). For more results related to the computational complexity of MAXRTC, see [8].

Motivation. The existing approximation algorithms for MAXRTC typically produce trees with an arbitrary number of internal nodes. For example, the algorithms in [7, 8] always produce a tree with $n-1$ internal nodes and the algorithm in [10], $n-1$ for certain $\mathcal{R}$. However, due to the issue of spurious novel clades [3] 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 paper. 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. On a high level, q-MAXRTC is related 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 [4, 5], 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 be 1 trillion [17]), to make their comparison practical, it may become necessary to approximate them using trees with fewer internal nodes while keeping enough triplet branching structure to represent them accurately.

New Results and Outline of the Article. Section 2 shows that q-MAXRTC is NP-hard for every fixed $q \geq 2$ and gives some inapproximability results. Section 3 describes our new approximation algorithms. Section 4 provides implementations and our experimental results. Section 5 presents a new algorithm for computing the rooted triplet distance between two trees. Finally, Section 6 contains some open problems. For a summary of previous and new results related to q-MAXRTC refer to the table below. Due to space constraints, some proofs and experimental results have been deferred to the journal version.

Year	Reference	Deterministic	q	Approximation	Type
1999	Gąsieniec et al. [10]	yes	unbounded	1/3	absolute
2010	Byrka et al. [7, 8]	yes	$n-1$	1/3	absolute
2019	new [Section 3.1]	no	2	1/2	relative
2019	new [Section 3.1]	yes	2	1/4	relative
2019	new [Theorem 7]	yes	2	4/27	absolute
2019	new [Theorem 9]	yes	$q \geq 3$	$1/3 - 4/(3(q + q \bmod 2)^2)$	absolute

2 Computational Complexity of q-MAXRTC

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

▶ **Theorem 1.** *q-MAXRTC is NP-hard for every fixed $q \geq 2$.*

Proof. We consider the known NP-hard problem MAX q-CUT [15], in which the input is an undirected graph $G = (V, E)$ and the goal is to find a partition $(A_1, A_2, \dots, A_q)$ of V such that the total number of edges connecting two nodes 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 $(A_1, A_2, \dots, A_q)$ 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.

First, assume that there exists a cut $(A_1, A_2, \dots, A_q)$ of size k in G. We construct a tree T that is rooted at the vertex a_1 with additional internal nodes $a_2, \dots, a_q$ such that a_{i+1} is a child of a_i for $1 \leq i \leq q-1$. For $i \in \{1, 2, \dots, q\}$, we attach $|A_i|$ leaves bijectively labeled by A_i as children of a_i, and the vertex 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 $i, j \in \{1, 2, \dots, q\}$. 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 will be consistent with T, so T will be consistent with exactly k triplets from $\mathcal{R}$.

Conversely, assume that there exists a tree T with q internal nodes $a_1, a_2, \dots, a_q$ that is consistent with k triplets from $\mathcal{R}$. Let $A_i = \{x : x \text{ is a child of } a_i\} \setminus \{z, a_1, a_2, \dots, a_q\}$ for $1 \leq i \leq q$. Define $S = \mathcal{R} \cap rt(T)$. For each $xz|y \in S$, clearly x and y belong to different sets A_i and A_j for some $i, j \in \{1, 2, \dots, q\}$, 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 [11], we obtain the following corollary:

▶ **Corollary 2.** *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 [15], we obtain the following corollary:

▶ **Corollary 3.** *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 $\epsilon > 0$.*

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 that can be consistent with a tree T with q internal nodes.

▶ **Lemma 4.** *Let $2 \leq q' \leq q \leq n-1$. We have that $opt(q') \leq opt(q) \leq \lceil \frac{q-1}{q'-1} \rceil opt(q')$.*

Proof. We start by showing that $opt(q') \leq opt(q)$. Let T' be the 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, let $u \in T$ such that $deg(u) > 2$ and u_1, u_2 be two children of u. Create an internal node u_{12}, make u_1 and u_2 the children of u_{12} and u_{12} the child of u.

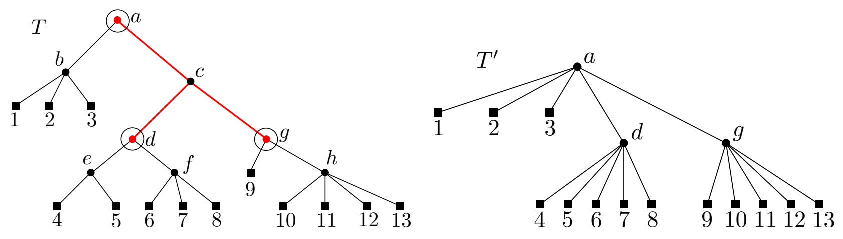

Figure 2 An example. Let T be the tree on the left with 9 internal nodes. The tree T' on the right with 3 internal nodes is created by deleting all internal nodes in T except $W = \{a, d, g\}$.

To show the second half 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 the tree that induces $opt(q)$ triplets from $\mathcal{R}$. Let $t = ab|c$ be a triplet induced in T that is also in $\mathcal{R}$. Anchor t in $lca(a, b)$. Let $W = \{u_1, u_2, \ldots, u_{q'}\}$ be any set of q' internal nodes in T such that the root of T is included in W. Create a tree T' with q' internal nodes by letting T' be a copy of T and applying the delete operation to every internal node of T' not in W. Note that for a node u in T such that $u \in W$, every triplet anchored in u will also be induced by T'. An example can be found in Figure 2.

Let $T'_1, T'_2, \ldots, T'_\lambda$ be trees that are built like T', but in a way such that every internal node $u \in T$ except $r(T)$, corresponds to an internal node of exactly one such tree. Observe that $\lambda = \lceil \frac{q-1}{q'-1} \rceil$.We can create these trees with the following procedure:

- Store all internal nodes of T except $r(T)$ in the ordered set S, in any order from left to right and set $j = 1$.
- If $|S| \geq q' - 1$, pick and remove from S the first $q' - 1$ internal nodes to define W, and construct T'_j. Otherwise, pick the remaining nodes in S to define W and create T'_j just like T' but with $|S| = |W|$ nodes instead of $q' - 1$. Set $j = j + 1$.
- if $|S| = 0$ stop. Otherwise go to the previous step.

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

3.1 Approximation Algorithms Based on MAX 3-CSP

In this subsection, we consider polynomial-time approximation algorithms. 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 [22], 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 5.** *If MAX 3-AND can be approximated within a factor of r, then 2-MAXRTC can also be approximated within a factor of r.*

Lemma 5 allows every approximation algorithm for MAX 3-AND to be used to approximate 2-MAXRTC. For MAX 3-AND, Zwick [22] presented a randomized $\frac{1}{2}$-approximation algorithm with relative ratio based on semi definite programming. Trevisan [19] presented a deterministic $\frac{1}{4}$-approximation algorithm with relative ratio based on linear programming. A deterministic algorithm based on local search by Alimonti [2], 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 4 this algorithm also gives a $\frac{1}{8}$-approximation ratio for q-MAXRTC, where $q \geq 3$.

3.2 Approximation Algorithms Based on Derandomization

This subsection also assumes that all approximation algorithms run in polynomial time. Reducing 2-MAXRTC to MAX 3-AND can produce a deterministic $\frac{1}{8}$-approximation algorithm for q-MAXRTC, however from Lemma 4, 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 4. We propose a new deterministic 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 the only available related algorithm based on derandomization by Byrka et al. [7], always constructs a binary tree on n leaves, i.e. the case $q < n-1$ is not considered. Moreover, as we will show below, our derandomization procedure is highly optimized for trees instead of the more complex *phylogenetic networks* (for a definition see Section 2 of [7]).

▶ **Lemma 6.** *There exists a randomized $\frac{4}{27}$-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 7.** *There exists a deterministic $\frac{4}{27}$-approximation algorithm for q-MAXRTC that runs in* $O(|\mathcal{R}|)$ *time.*

Proof. We derandomize the algorithm in Lemma 6 with the method of conditional expectations [20] in a way that differs from Byrka et al. [7], 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 in a way, such that after every assignment the expected value of the solution is preserved. 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 N_i contain those assignments, i.e., $N_i = \{x_1 \leftarrow n_1, \ldots, x_i \leftarrow n_i\}$. 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() of Algorithm 1). A trivial implementation that scans 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 $\mathcal{R}$ would require $O(n|\mathcal{R}|)$ time.

We can achieve a more efficient implementation (see procedure 2-MAXRTC-FAST() of Algorithm 1) that would require $O(|\mathcal{R}|)$ time, by maintaining for every leaf $x_i \in L$, a list of all the triplets that x_i is part of, denoted $\mathcal{R}[x_i]$. At the beginning of the i-th iteration of the first for loop in Algorithm 1, 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, 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

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

```
1: procedure PR2($xy|z$)                                    ▷ Computing $Pr[xy|z \in rt(T)|N_i]$
2:     if $x \leftarrow a$ or $y \leftarrow a$ or $z \leftarrow b$ then return 0
3:     $p = 4/27$
4:     if $x \neq \emptyset$ and $x \leftarrow b$ then $p = 3p/2$      ▷ $x \neq \emptyset$ meaning that $x$ has been assigned
5:     if $y \neq \emptyset$ and $y \leftarrow b$ then $p = 3p/2$
6:     if $z \neq \emptyset$ and $z \leftarrow a$ then $p = 3p$
7:     return $p$
8: procedure 2-MAXRTC-FAST($\mathcal{R}$)                          ▷ The main procedure
9:     $prev = 4|\mathcal{R}|/27$                           ▷ Storing $E[W|N_0]$, where $N_0 = \emptyset$
10:    for $i = 1$ to $n$ do
11:        $aValue = prev$                                  ▷ To compute $E[W|N_{i-1}, x_i \leftarrow a]$
12:        $bValue = prev$                                  ▷ To compute $E[W|N_{i-1}, x_i \leftarrow b]$
13:        for $j = 1$ to $|\mathcal{R}[x_i]|$ do
14:            $x_i \leftarrow \emptyset$
15:            $aValue = aValue - \mathrm{PR2}(\mathcal{R}[x_i][j])$
16:            $bValue = bValue - \mathrm{PR2}(\mathcal{R}[x_i][j])$
17:            $x_i \leftarrow a$
18:            $aValue = aValue + \mathrm{PR2}(\mathcal{R}[x_i][j])$
19:            $x_i \leftarrow b$
20:            $bValue = bValue + \mathrm{PR2}(\mathcal{R}[x_i][j])$
21:        $x_i \leftarrow b$
22:        $prev = bValue$
23:        if $aValue > bValue$ then
24:            $x_i \leftarrow a$
25:            $prev = aValue$
```

to the value of *prev*, and then for every triplet in the list $\mathcal{R}[x_i]$, 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 3 lists, every triplet will induce $O(1)$ calls to the procedure PR2() of Algorithm 1, giving the $O(|\mathcal{R}|)$ final bound of the algorithm. ◀

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 7 optimal when considering algorithms with absolute approximation ratios.

▶ **Theorem 8.** *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 $|\mathcal{R}_n| = 3\binom{n}{3}$. Next, we construct a tree T with two internal nodes, which is rooted at the vertex a with an internal node b (b is a child of a). Let $A = \{x : x \text{ is a child of } a\} \setminus \{b\}$ and $B = \{x : x \text{ is a child of } b\}$. Assume that $m = |A|$. Then $|B| = n - m$ 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 $\lim\limits_{n\to\infty} \frac{|rt(T)\cap\mathcal{R}_n|}{|\mathcal{R}_n|} = \frac{4}{27}$. ◀

To obtain an algorithm that has a better approximation ratio for $q \geq 3$, 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 6. Let $E[W]$ be the expected value of that randomized algorithm. Like in Theorem 7, 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 $E[W] > \frac{4}{27}|\mathcal{R}|$.

▶ **Theorem 9.** *Given $q \geq 3$, there exists a randomized 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* $\mathrm{O}(q|\mathcal{R}|)$.

4 Implementation and Experiments

We used the C++ programming language to implement the algorithm from Theorem 7 for 2-MAXRTC, and the algorithm from Theorem 9 for q-MAXRTC when $q > 2$. The implementation is publicly available at `https://github.com/kmampent/qMAXRTC`. Below, we describe some experiments on both simulated and real datasets and the results.

Simulated Dataset. The input to q-MAXRTC is a set of triplets $\mathcal{R}$ and a parameter q. We define the following types of sets for $\mathcal{R}$:

- *dense consistent* (abbreviated `dc`): if $|\mathcal{R}| = \binom{n}{3}$ and $\mathcal{R}$ is consistent with a tree T containing $n-1$ internal nodes. The tree T is created using the uniform model [18].
- *probabilistic*: if $|\mathcal{R}| = n^2$ and $\mathcal{R}$ is a set of triplets on n leaf labels created as follows. After building a binary tree T on n leaves following the uniform model, start extracting triplets from T to add into $\mathcal{R}$. For every extracted triplet $xy|z$, permute the 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.

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. Figure 3 compares the performance of q-MAXRTC, `WU`, and `OLS` in the `dc` and `prob50` models, for small values of q and n at most 1000. In both models, the larger the value of q, the better the performance of q-MAXRTC. Moreover, the improvement in performance decreases as the value of q increases, which is expected. For the `dc` dataset, which contains no conflicting triplets, the performance is much better. Significantly, when $q = 9$ we can capture close to 80% of the triplets even if the input tree contains as many as 1000 leaves. When compared against `WU` & `OLS`, we can see that while `WU` & `OLS` perform better, the difference in performance is small compared to the difference in the number of internal nodes used by the algorithms.

Real Dataset. We considered five trees from recently published papers ([9] and [16]). From [9] we used the trees from the supplementary datasets 2 and 4, denoted `nmS2` and `nmS4` respectively. From [16] 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,

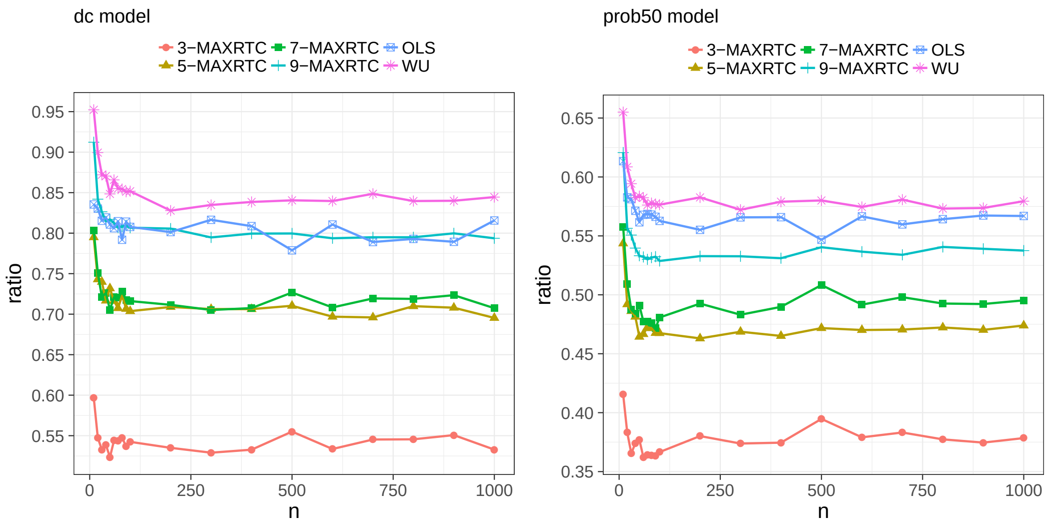

Figure 3 Performance of $\{3, 5, 7, 9\}$-MAXRTC compared to `WU` and `OLS` in the `dc` and `prob50` models. Every data point corresponds to the mean of 100 runs. Observe that the performance of 9-MAXRTC is very close to that of `WU` & `OLS`, even though 9-MAXRTC uses only 9 internal nodes, while `WU` uses exactly $n - 1$ internal nodes and `OLS` at most $n - 1$.

Table 1 Performance of q-MAXRTC on real datasets. Every cell corresponds to the best ratio (as defined below) 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

we removed the two leaves that are 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. Let $D(T, T_q)$ be the rooted triplet distance between T and T_q (for a definition see Section 5 below). The performance 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 [5]. We measured the performance 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 1 shows the best ratios achieved, and the corresponding trees in Newick format can be found at `https://github.com/kmampent/qMAXRTC`. As can be seen from the results, larger number of internal nodes tend to improve performance. Significantly, with only 9 nodes we can capture between 71% and 86% of the triplets in each case, and with 11 nodes between 79% and 91%. When $q > 11$, we did not observe a significant improvement in performance.

5 Motivation for q-MAXRTC: Faster Computation of 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.

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 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. Similarly to a resolved triplet, 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 $D(T_1, T_2) = \binom{n}{3} - S(T_1, T_2)$.

The Algorithm. It is known how to compute $D(T_1, T_2)$ in $O(n \log n)$ time [4, 5]. Below, we show how to compute $D(T_1, T_2)$ in $O(qn)$ time, which is faster than [4, 5] when $q = o(\log n)$. There is 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 a 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 $C[u][r] - C[u][l - 1]$.

Counting. We extend the technique introduced in [5]. 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 10.** *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.*

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 capture 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 every other leaf in T_2 have the color white. The red, blue, and green colors will be used to capture fan triplets and the red, blue, and white colors, resolved triplets. By the relabeling scheme of the leaves, we have that the red, blue, and green colors 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. Note that $a_{\text{blue}} = a'_{\text{red}} + 1$ 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 are now going to 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 $S_r^{\{v,v'\}}(T_1, T_2)$ denote the shared resolved triplets anchored in $\{v, v'\}$ and similarly let $S_f^{\{v,v'\}}(T_1, T_2)$ denote the shared fan triplets. Note that we have $S^{\{v,v'\}}(T_1, T_2) = S_r^{\{v,v'\}}(T_1, T_2) + S_f^{\{v,v'\}}(T_1, T_2)$. The following lemma gives an algorithm for computing $S^{\{v,v'\}}(T_1, T_2)$ efficiently.

▶ **Lemma 11.** *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 a $\mathrm{O}(q)$-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 $T(I) = \{T(u) : u \in I\}$. 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 $(i, j) \in \{(red, blue), (red, green), (blue, green)\}$: 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.

To compute these counters for every internal node of T_1 efficiently, a depth first traversal is applied on T_1 while making sure that we only visit internal nodes. For every such internal node u visited, a simple dynamic programming procedure is used to compute the counters of u in $\mathrm{O}(|I|)$ time, thus making the total time required to compute all counters $\mathrm{O}(q)$.

Algorithm 2 shows how to compute $S_f^{\{v,v'\}}(T_1, T_2)$ and $S_r^{\{v,v'\}}(T_1, T_2)$ in $\mathrm{O}(q)$ time as well. It 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 $T(I)$, (2) two leaves come from $T(I)$ and one from L, (3) one leaf comes from $T(I)$ and two from L, and (4) all three leaves come from L. Similarly, for the leaves of a resolved triplet we have the following cases: (1) two leaves come from $T(I)$ and one not from $T_1(u)$, (2) one leaf comes from $T(I)$, one from L, and one not from $T_1(u)$, and (3) two leaves come from L and one not from $T_1(u)$. 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 statement follows. ◀

Algorithm 2 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
```

Algorithm 3 $O(qn)$-time algorithm for computing $D(T_1,T_2)$.

```
1: procedure $D(T_1,T_2)$
2:     Compute the $q \times n$ table $C$.
3:     For every $u$ in $T_1$ compute the parameter $u_l$, which is the number of leaves in $T(u)$.
4:     sharedTriplets = 0
5:     for every internal node $v$ in $T_2$ 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:             Given $C$, $R$, $B$, and $G$, compute the counters of $T_1$ according to Lemma 11
9:             sharedTriplets = sharedTriplets + $S_f^{\{v,v'\}}(T_1,T_2) + S_r^{\{v,v'\}}(T_1,T_2)$
10:    return $\binom{n}{3}$ − sharedTriplets
```

In Algorithm 3 we show how to compute $D(T_1,T_2)$. 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 11, 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 10, thus we obtain the following theorem:

▶ **Theorem 12.** *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.*

An implementation of the algorithm in C++ is available at `https://github.com/kmampent/qtd`. Preliminary 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 [5] for large inputs, e.g., when $n = 1,000,000$ and $q \leq 50$. Details will be reported in the full version of the paper.

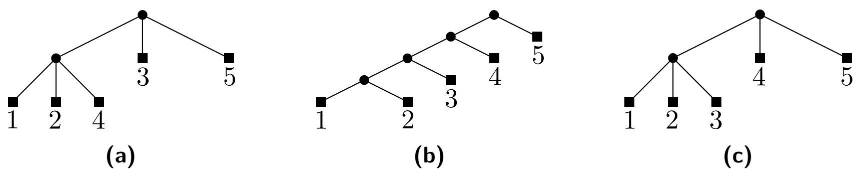

Figure 4 Let $\mathcal{R} = \{12|3, 13|4, 24|5\}$. (a) The optimal tree for 2-MAXRTC induces 2 triplets from $\mathcal{R}$. (b) The tree returned by the BUILD algorithm from [1]. (c) The best tree obtainable by contracting all internal edges except one in the tree from (b) induces only 1 triplet from $\mathcal{R}$, so this method is not optimal for 2-MAXRTC.

6 Open Problems

The optimal polynomial-time approximation ratio for any fixed $q \geq 3$ is an open problem, as well as the existence of algorithms achieving that ratio. Moreover, for the special case where all the triplets in $\mathcal{R}$ are consistent with a tree T, the computational complexity of q-MAXRTC is an open problem as well. Note that just applying BUILD [1] to obtain such a T and then trying every bipartition of L induced by an edge of T fails to produce an optimal solution to 2-MAXRTC (see Figure 4 for a counterexample). Another open problem is the existence of approximation algorithms for q-MAXRTC in the weighted case, where each triplet in $\mathcal{R}$ has a weight and the objective is to build a tree that maximizes the total weight of the triplets induced from $\mathcal{R}$. This addresses the case where some triplets in $\mathcal{R}$ are more important than others. Moreover, another open problem is the following: given a set of triplets $\mathcal{R}$ on a leaf label set of size n and a parameter ℓ, build 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* studied in [13] and MAXRTC. Finally, for the rooted triplet distance computation, a major open problem [4, 5] is whether it can be computed in $\mathrm{O}(n)$ time. When $q = \mathrm{O}(1)$, our proposed algorithm runs in $\mathrm{O}(n)$ time. If q_1 is the total number of internal nodes of one tree and q_2 of the other, is it possible to obtain an algorithm with a $\mathrm{O}(q_1 q_2 + n)$ running time?

References

1 A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

2 P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. *Information Processing Letters*, 57(3):151–158, 1996.

3 O. R. P. Bininda-Emonds. The evolution of supertrees. *Trends in Ecology & Evolution*, 19(6):315–322, 2004.

4 G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, T. Mailund, and A. Sand. Efficient Algorithms for Computing the Triplet and Quartet Distance Between Trees of Arbitrary Degree. In *Proc. SODA 2013*, pages 1814–1832, 2013.

5 G. S. Brodal and K. Mampentzidis. Cache Oblivious Algorithms for Computing the Triplet Distance between Trees. *Proc. ESA 2017*, pages 21:1–21:14, 2017.

6 D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees - Theory and Methods in Phylogenetic Analysis*. PhD thesis, University of Canterbury, Christchurch, NZ, 1997.

7 J. Byrka, P. Gawrychowski, K. T. Huber, and S. Kelk. Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks. *Journal of Discrete Algorithms*, 8(1):65–75, 2010.

8 J. Byrka, S. Guillemot, and J. Jansson. New Results on Optimizing Rooted Triplets Consistency. *Discrete Appl. Math.*, 158(11):1136–1147, 2010.

9 L. A. Hug et al. A new view of the tree of life. *Nature Microbiology*, 1, 2016.

10 L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Constructing Evolutionary Trees. *Journal of Combinatorial Optimization*, 3(2):183–197, 1999.

11 Johan Håstad. Some Optimal Inapproximability Results. *J. ACM*, 48(4):798–859, 2001.

12 J. Jansson, R. S. Lemence, and A. Lingas. The Complexity of Inferring a Minimally Resolved Phylogenetic Supertree. *SIAM Journal on Computing*, 41(1):272–291, 2012.

13 J. Jansson, Joseph H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted Maximum Agreement Supertrees. *Algorithmica*, 43(4):293–307, 2005.

14 J. Jansson, R. Rajaby, and W.-K. Sung. Minimal Phylogenetic Supertrees and Local Consensus Trees. *AIMS Medical Science*, 5:181, 2018.

15 V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the Hardness of Approximating MAX k-CUT and Its Dual. *Chicago Journal of Theoretical Computer Science*, 1997.

16 J. M. Lang, A. E. Darling, and J. A. Eisen. Phylogeny of bacterial and archaeal genomes using conserved genes: supertrees and supermatrices. *PLoS ONE*, 8(4), 2013.

17 K. J. Locey and J. T. Lennon. Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences*, 2016.

18 A. McKenzie and M. Steel. Distributions of cherries for two models of trees. *Mathematical Biosciences*, 164(1):81–92, 2000.

19 L. Trevisan. Parallel Approximation Algorithms by Positive Linear Programming. *Algorithmica*, 21(1):72–88, 1998.

20 D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*, pages 108–109. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

21 B. Y. Wu. Constructing the Maximum Consensus Tree from Rooted Triples. *Journal of Combinatorial Optimization*, 8(1):29–39, 2004.

22 U. Zwick. Approximation Algorithms for Constraint Satisfaction Problems Involving at Most Three Variables Per Constraint. *Proc. SODA 1998*, pages 201–210, 1998.