RogueNaRok: an Efficient and Exact Algorithm for Rogue Taxon Identification

Andre J. Aberer*[1], Denis Krompaß[1] and Alexandros Stamatakis[1]

Heidelberg Institute for Theoretical Studies (HITS gGmbH),
Schloss-Wolfsbrunnenweg 35, D-69118 Heidelberg
{Andre.Aberer,Denis.Krompass,Alexandros.Stamatakis}@h-its.org
WWW home page: http://exelixis-lab.org

Abstract. The presence of *rogue taxa* (rogues) in a tree set can have detrimental effects on the quality of consensus trees constructed from such a set. We introduce a graph-based algorithm for rogue taxa identification that can accurately assess the improvement induced by removing subsets of taxa from the set of trees. The new algorithm is up to four orders of magnitude faster than our previous algorithm while returning exactly identical results. Our approach is capable of identifying rogues in tree sets with more than 110,000 taxa. On a large collection of real-world datasets, we show that, identifying and pruning rogues with our new algorithm yields more informative results than a competing approach.

Keywords: phylogenetic post-analysis, rogue taxa, consensus tree

1 Introduction

1.1 Phylogenetic Analysis And Rogue Taxa

The goal of phylogenetic analyses is to infer the evolutionary relationships between a set of species (also called taxa) which are usually represented by a tree. As input, genetic data (i.e., protein or DNA sequences) in form of a multiple sequence alignment (MSA) are used. Popular approaches for MSA-based phylogenetic inference are Maximum Parsimony (MP), Maximum Likelihood (ML), and Bayesian methods (reviewed in [6]).

For MP and ML, we can assess the support for taxonomic relationships by randomly re-sampling MSA sites to generate bootstrap replicates [5]. Then, so-called bootstrap trees are computed on those replicates. When the tree space is sampled using Bayesian methods one also obtains a tree collection. Usually, a consensus tree is built to summarize the information contained in those tree sets. However, the resolution in a consensus tree can be substantially decreased by *rogues* (the term rogue/rogue taxa was introduced in [19, 20, 18]), that assume varying and often contradictory positions in a tree set. The rogue phenomenon is usually attributed to ambiguous or insufficient phylogenetic signal [10].

* topic suggestion: molecular evolution

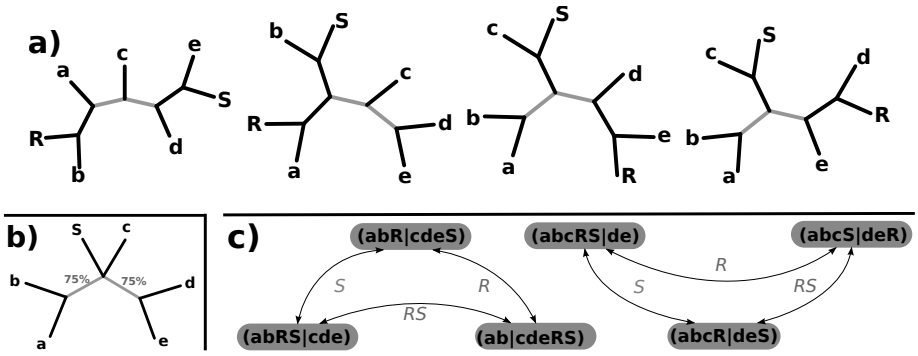

Fig. 1: *a)* four trees with seven taxa including two rogue taxa R and S. Bipartitions recoverable in an MRC tree are indicated in gray. *b)* MRC tree after pruning R. *c)* The two connected components $\chi[\{R, S\}]$ for the dropset $\{R, S\}$ of the four initial trees.

Determining the "correct" position of a rogue in a phylogenetic tree is tedious [11] and therefore rogues, once identified, are mostly just excluded (pruned) from studies. Criteria based on triple frequencies [17] or node distances [8] are often applied [16, 15, 4, 13] to identify rogues. Recently, Pattengale *et al.* introduced a fast method that approximates the potential increase of resolution in the consensus tree when rogues are pruned [9]. We refer to this algorithm as *bipartition merging algorithm* (BMA). Our exact, but significantly slower, *single-taxon algorithm* (STA) [2] assesses the inherent support improvement induced by pruning one taxon at a time. We showed that, the STA and BMA consistently identify rogues with a more harmful effect on the consensus tree support than rogues identified by triple frequency or node distance methods.

Here, we present RogueNaRok, an exact and efficient algorithm to determine the effects (on consensus support) of pruning rogues that is available at `https://github.com/aberer/RogueNaRok`. The remainder of this paper is organized as follows. Initially, we formalize the rogue problem as optimization problem (Section 1.2). Then, we introduce a generalized graph-based algorithm and an approximation (Section 2) for rogue identification. We discuss algorithmic and technical optimizations of the implementation in Section 3. We compare runtimes and result quality to competing methods in Section 4 and explore the scalability (in terms of tree size) of our implementation. Finally, we conclude and address directions of future work (Section 5).

1.2 Problem Description

Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be a collection of m unrooted phylogenetic trees, that is, binary trees which are labeled at degree-1 nodes with labels $L = \{l_1, \ldots, l_n\}$ (see Fig. 1). Each inner branch B_i of each tree splits L into b_i and $\bar{b_i}$ (with $|b_i| > 1$ and $|\bar{b_i}| > 1$) yielding a set of bipartitions $\mathcal{B} = \{B_1, \ldots, B_j\}$, where $B_i = (b_i|\bar{b_i})$. The function $\sigma : \mathcal{B} \rightarrow 2^{\mathcal{T}}$ maps a bipartition $B_i \in \mathcal{B}$ to the set

of trees in which B_i occurs. For the purpose of constructing a consensus tree, the *bipartition profile* ($\mathcal{B}$, σ) is equivalent to and contains the same information as $\mathcal{T}$.

The consensus tree for a given bipartition profile $(\mathcal{B}, \sigma)$ can be defined as $C_t(\mathcal{B}) = \{B \in \mathcal{B}, \text{s.t. } |\sigma(B)| > t\}$, where t is a threshold between $\frac{m}{2}$ (majority rule consensus (MRC)) and m-1 (strict consensus (SC)). As optimality criterion for consensus trees in the rogue identification context, we can use the resolution of a consensus tree $|C_t(\mathcal{B})|$ (e.g., the *relative information content* (RIC) [9]) or the sum over bipartition support (bipartition frequencies) in a consensus tree $\sum_{B \in C_t(\mathcal{B})}(|\sigma(B)|)$ (e.g., the *relative bipartition information content* (RBIC) measure [2]).

Fig. 1 provides an example of the negative impact of rogues on the RIC or RBIC score of a consensus tree. For a MRC threshold, the consensus tree of the four bootstrap trees does not contain a single bipartition, thus resolution and cumulative support are 0. Pruning rogue R from the bootstrap trees yields (resp. *recovers*) 2 out of 3 possible consensus bipartitions for a resolution-based criterion. An algorithm optimizing a support-based criterion, will also prune taxon S, which yields two bipartitions with a support of 100% each, instead of 75%. Both basic optimality criteria (RIC and RBIC) can be modified to penalize the loss of taxa, since in Fig. 1 the partially resolved position of S also contains information. While this influences the number of rogues pruned by an algorithm, it does not have further implications for algorithm design or complexity. Note that, for an SC tree of Fig. 1, we only obtain an improvement, if we prune R and S simultaneously.

If we prune a set of taxa d (referred to as *dropset*) from a set of bipartitions $\mathcal{B}$ (resp. the underlying trees), we obtain pruned bipartitions $\mathcal{B}_{-d}$. The task of rogue identification is to compute an optimal dropset d_{opt}, that maximizes the optimality criterion used (such as RBIC or RIC) for the given input tree collection. A consensus tree that is optimal with respect to the optimality criterion is called *Maximum-Information Subtree Consensus* (MISC) [9].

2 Algorithm

2.1 Motivation

BMA and STA are greedy algorithms, that is, as long as improvement is possible, they search for a dropset inducing the best improvement of resolution/support. They prune this dropset, recompute the bipartition profile, and then search for the next best dropset again. BMA searches for dropsets of arbitrary size to increase resolution. While it is particularly fast, this approximation occasionally chooses dropsets that are sub-optimal or even decrease resolution. As an alternative, we introduced STA, that determines the exact optimality change of pruning only one taxon at a time (dropsets of size 1) and thereby greedily optimizes the accumulated support of a consensus tree. While this strategy performed well for rogue detection, runtimes become prohibitive on large datasets with more than

Algorithm 1 maps all minimal dropsets to induced optimality change

Input: a set of bipartitions $\mathcal{B}$
Output: optimality change s_d for each minimal dropset d

function createCombineScoreDropsets($\mathcal{B}$)
  $D \leftarrow \{d \mid \exists (B_i, B_j) \in \mathcal{B}^2, (b_i \Delta b_j = d) \vee (b_i \Delta \bar{b_j} = d)\}$ ▷ I. compute dropsets
  for all $d \in D$ **do**
    $\varepsilon[d] \leftarrow \{(B_i, B_j) \in \mathcal{B}^2 \mid (b_i \Delta b_j = d) \vee (b_i \Delta \bar{b_j} = d)\}$
  end for
  for all $d \in D$ **do** ▷ II. compute partial graphs
    $\varepsilon^*[d] \leftarrow \{(B_i, B_j) \in \mathcal{B}^2 \mid (B_i, B_j) \in \varepsilon[d'] \wedge d' \subseteq d\}$
    $\chi[d] \leftarrow$ connectedComponents($\varepsilon^*[d]$)
  end for
  for all $d \in D$ **do** ▷ III. evaluate dropsets
    $s_d \leftarrow$ calculateOptimalityChange($d, \chi, \mathcal{B}$)
  end for
end function

1,000 taxa. STA repeatedly hashes bipartitions to implicitly determine mergers between them. Thus, a large amount of time is spent to (re-)calculate hash values of bipartitions that do not change, if a taxon is pruned. Moreover, we observed that, the optimality change for many dropsets remains constant over several iterations of the algorithm, that is, over several pruning steps.

2.2 General Algorithm

We introduce a graph-based formulation of the rogue problem and design an algorithm, that exactly determines how the bipartition profile changes, when dropsets are pruned from the underlying trees. Bipartitions stored in a bipartition profile can change in two ways when taxa are pruned: (i) a bipartition vanishes, because it degenerates into a trivial (non-informative) bipartition (i.e., $(|b_i| < 2) \vee (|\bar{b_i}|) < 2)$; (ii) a set of bipartitions $\{B_i, \ldots, B_j\}$ merges into a new bipartition B' with support $\sigma(B') = \sigma(B_i) \cup \ldots \cup \sigma(B_j)$. If a consensus bipartition vanishes, this is detrimental for the optimality of the resulting consensus tree. Case (ii) can either increase or decrease the optimality, depending on the support of B' and $B_i, \ldots, B_j$.

For each pair of bipartitions (B_i, B_j) there exist two dropsets $d = b_i \Delta b_j$ and $\bar{d} = \bar{b_i} \Delta b_j$ that induce a merger between the pair, where $a \Delta b$ is the symmetric difference $(a \cup b) \setminus (a \cap b)$ (for details see [9]). Dropsets d and $\bar{d}$ are called *minimal* when the following property holds: If we remove any taxon from d or $\bar{d}$, then bipartitions B_i and B_j do not merge. Bipartitions $\mathcal{B}$ and the minimal dropset d and $\bar{d}$ for each pair in $\mathcal{B}^2$ can be represented as *merging graph* G of possible mergers. Each bipartition represents a node and each pair of nodes is connected by two directed edges (labeled with d and $\bar{d}$). The edge indicates that this pair of bipartitions merges, if d (resp. $\bar{d}$) is pruned.

Based the merging graph G, we can now formulate Alg. 1) that iterates over all possible minimal dropsets and determines the consensus tree optimality score

for pruning the dropset from the underlying trees. The algorithm consists of three phases: In phase I, we determine all minimal dropsets and create a mapping ε, that maps each dropset d to edges in the merging graph G labeled with d. In phase II, for each dropset d, we gather all edges of G that are labeled with either d or a subset of d, thus creating ε^*. Each non-singleton connected component in the graph G_d induced by the node set $\mathcal{B}$ and the edge list $\varepsilon^*[d]$ describes a set of bipartitions $\{B_i, \ldots, B_j\}$ that merge into a new bipartition B', when d is pruned. Fig. 1 c) illustrates the connected components of a merging graph $G_{\{R,S\}}$ for the two bipartitions that can be recovered in the trees of Fig. 1 a).

Finally, in phase III, we calculate the optimality change s_d induced by each minimal dropset d (CALCULATEOPTIMALITYCHANGE in Alg. 1). Here, we use support-based optimality (RBIC). Using resolution-based optimality (RBIC) instead is straight-forward. The bipartitions of each connected component $c = \{B_i, \ldots, B_j\} \in \chi[d]$ merge into a bipartition B', if d is pruned. Let $\delta(a,t) = a$, if $a > t$ and 0 otherwise, where t is the consensus threshold. Then the optimality change s_d of d is

$$s_d = \sum_{c \in \chi[d]} \left(\overbrace{\delta(|\sigma(B')|, t)}^{s_{gain}(c)} - \overbrace{\sum_{B_k \in \{B_i, \ldots, B_j\}} \delta(|\sigma(B_k)|, t)}^{s_{loss}(c)} \right) - \overbrace{\sum_{B_l \in B^{cons'}} |\sigma(B_l)|}^{s_{van}(d)}, \quad (1)$$

where in Equation 1 $s_{gain}(c)$ is the support attained by the new bipartition B' and $s_{loss}(c)$ the support lost by the merging bipartitions. The term $s_{van}(d)$ accounts for consensus bipartitions ($B^{cons'}$) that degenerate into trivial bipartitions and do not occur in any connected component.

CREATECOMBINESCOREDROPSETS computes the exact optimality score change for each minimal dropset. We can use this algorithm to repeatedly determine the best dropset (most support recovered per taxon pruned) and then transform the bipartition profile accordingly, until no further optimality score improvement is possible. The algorithm is polynomial in the number of bipartitions $|\mathcal{B}|$, since all phases are polynomial (including the search for connected components [7]). Note that, this algorithm is still a greedy approximation to the globally optimal MISC. This is because of the term s_{van} in the dropset evaluation. For the terms s_{loss} and s_{gain}, the merger graph implies that two dropsets d_1 and d_2 will either not influence each other's difference $s_{gain} - s_{loss}$ or, that there exists a dropset $d_1 \cup d_2$ that will be evaluated at some point. The term s_{van} however, needs to be evaluated for all possible combinations of minimal dropsets 2^D. Thus, we suspect that the *MISC*-problem may be $\mathcal{NP}$-hard.

Phase III of the algorithm can be modified to optimize the bootstrap support of bipartitions that are drawn on a best-known ML reference tree. Note that, if bipartitions $\{B_i, \ldots, B_j\}$ merge into B', then B' forms part of the ML reference tree, if at least one bipartition in $\{B_i, \ldots, B_j\}$ occurred in the ML tree. We can also use the merger information in $\chi[d]$ for the optimization of support in an extended majority rule (MRE) tree. This is computationally expensive, since we have to compute a MRE tree for each possible minimal dropset d from a bipartition profile that has been transformed according to $\chi[d]$.

2.3 Approximation

There are $2 \cdot |\mathcal{B}|^2$ edges in the merger graph G and the subsequent search for sub-dropsets also has quadratic time requirements. Thus, the general algorithm described in Section 2.2 quickly becomes prohibitive on real-world datasets. For example, consider that 1,000 trees with 1,000 taxa can contain almost $1,000,000$ different bipartitions in the worst case. Problematic real-world datasets of this size usually yield between 50,000 and 100,000 distinct bipartitions. Many edges in the merger graph are labeled by excessively large dropsets. For instance, if for two bipartitions B_i and B_j their dropset $d = b_i \Delta b_j$ is of size $|d| = 1$, then the complementary dropset $\bar{d}$ has size $|\bar{d}| = n - 1$ ($n :=$ number of taxa). This in fact means, that it is impossible for this dropset $\bar{d}$ to recover a bipartition.

In previous experiments on real datasets [2], we observed that, the BMA only prunes dropsets of size 1 in 90% of its iterations. Thus, we approximate and parametrize Alg. 1 by only computing the exact optimality score change for minimal dropsets of size $\leq l$. To implement this we need to modify line 2 in Alg. 1 as follows:

$$D \leftarrow \{d \mid \exists (B_i, B_j) \in \mathcal{B}^2, \big((b_i \Delta b_j = d) \vee (b_i \Delta \bar{b_j} = d)\big) \wedge |d| \leq l\} \tag{2}$$

If we only compute dropsets of size $l := 1$, not more than two bipartitions will merge into new bipartition at a time. This follows from the triangle inequality that holds for the lower-cardinality dropsets of a set of bipartitions $\{B_i, \ldots, B_j\}$ (which define a metric in the space of bipartitions [9]). As a consequence, for $l := 1$, we can omit the expensive phase II of Alg. 1 and $\forall d \in D : \chi[d] = \varepsilon[d]$.

3 Optimization and Implementation

3.1 Updating Instead of Recreating the Graph

Every iteration of Alg. 1 starts with the computation of minimal dropsets that define the edges in the merger graph. Thus, after we have selected dropset d in the previous iteration and transformed the bipartition profile as indicated by $\chi[d]$, we essentially recompute the edges of the merger graph from scratch. However, many of the recomputed edges will be identical to the edges of the merger graph in the preceding iteration. Let $d = b_i \Delta b_j$: if both bipartitions B_i and B_j did not merge with any other bipartition in the previous iteration and no taxa of the respective partitions b_i and b_j have been pruned, then the edge $(B_i, B_j) \in \varepsilon[d]$ remains unchanged. This means that in phase I of Alg. 1, we only need to exhaustively compute all edges of the merger graph for the first iteration. In subsequent iterations, we can simply update/modify the merger graph as required. Also, if an edge has not changed between iterations, we do not need to recompute s_{gain} and s_{loss} for the respective merging event. While we still have to compute s_{van}, this modification nonetheless accelerates phase III.

3.2 Implementation

We implemented the above algorithm in C. We represent bipartitions as bit vectors, because of their efficiency for set operations. Specifically, bits set in the bit vector for bipartition B_i denote the smaller partition b_i with $|b_i| \leq |\bar{b_i}|$. We sort our array of bipartitions by the number of bits that are set and create an index for rapidly accessing the first bipartition in an array of bipartitions with a specific number of set bits. We perform the necessary steps (inverting bit vectors representing the larger partition, sorting and indexing the bipartition array) at every iteration. This is done in an phase 0 that precedes phase I of Alg. 1.

The comparatively cheap operations in phase 0 allow to efficiently compute minimal dropsets in phase I, if the maximum dropset size is limited by parameter l as described in Section 2.3. A bipartition B_i with k set bits needs to be compared to all bipartitions B_j for which the smaller partition is of size x with $k-l \leq x \leq k+l$. All bipartitions for which the smaller partition has either more or less than x elements are not able to produce a dropset of size l with B_i. For $l := 1$, we only need to compare B_i to those bipartitions B_j in which either $k-1$ or $k+1$ bits are set.

We calculate the dropset for bipartitions B_i and B_j by applying the bit-wise *xor*-operator to the bit vectors. To save time, we discontinue the *xor*-operations on the bit vectors that typically comprise several bytes, when they already differ by more bits than specified by l.

We used the Pthreads-based fine-grained fork-join synchronization framework implemented in RAxML [14] to parallelize phases I through III. All three phases are essentially embarrassingly parallel. Only in phase I, we have to make hash table insertions thread-safe by using an appropriate locking mechanism.

4 Results

We executed the RogueNaRok algorithm (RNR), the STA and the BMA on collections of bootstrap trees from 34 real MSAs. If not stated otherwise, all tree sets contain 1,000 trees. The number of taxa ranges between 24 and 7,764. For details about the data sets and their availability, see [1, 2]. Runtime measurements were performed on unloaded AMD Magny-Cours nodes (8 processors with 6 cores each, 2.2 GHz, 128/256 GB RAM).

4.1 Runtime

Fig. 2 depicts execution times for the three algorithms. We executed the RNR algorithm for maximum dropset sizes $l := 1$ and $l := 2$ (referred to as RNR-1 and RNR-2). RNR-1 and the STA mostly produce identical results; results only differ for iterations where multiple dropsets are equally optimal. However, for all except the smallest datasets, RNR-1 is significantly faster than STA. Overall, we observe an average runtime improvement ranging between two and three orders of magnitude. In a case with 2,308 taxa and not more than 45,022

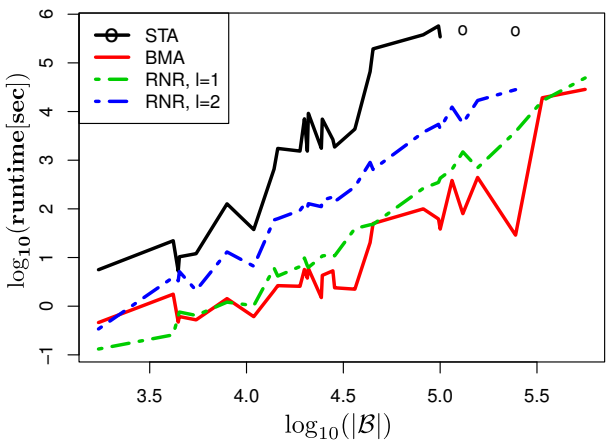

Fig. 2: Runtimes for the STA, BMA and RNR algorithm with maximum dropset size $l := 1$ and $l := 2$. x-axis refers to the initial number of bipartitions $|\mathcal{B}|$ for a bootstrap tree collection. Runtimes for MRC as consensus threshold (SC similar).

bipartitions the RNR algorithm is 4,047-fold faster than STA. The main reason for the significantly faster runtimes of RNR-1 is that we just carry over most edges of G into the next iteration, instead of computing G from scratch (as explained in Section 3.1). For instance, in the first iteration of the data set with 2,000 taxa, RNR spends 137 seconds in phase I to compute the minimal dropsets in G. In subsequent iterations, updating G takes between 0.05 and 10 seconds (mean: 1.2 seconds).

When choosing a larger l setting, collecting edges induced by sub-dropsets for the quadratically growing number of dropsets (phase II) starts dominating runtimes. Nevertheless, RNR-2 is in most cases still significantly faster than STA (see Fig. 2), while potentially being more accurate as well.

While RNR-2 is considerably slower than the BMA, RNR-1 achieves runtimes that are comparable to the BMA. However, the RNR algorithm usually identifies at least 10 times more rogues than the BMA. In terms of runtime per identified rogue, RNR-1 is faster than the BMA in all but two cases.

4.2 Parallel Scalability

We obtain up to 14-fold parallel speed-up (with 32 threads) for a maximum dropset size of $l := 1$. The reason for this sub-optimal speed-up is that, the parallel runtime is dominated by phase I during the initial iteration. As explained in Section 4.3, for $l > 1$, phase II starts dominating run-times. For $l := 2$, we observe a 7-fold parallel speed-up (with 48 threads) in the best case. Profiling of phase II indicates that dynamic memory allocation and deallocation on the heap consumes the largest fraction of time. We therefore further investigated the performance of the `malloc`-implementation. We replaced the default `malloc` by the Hoard memory allocator [3], but did not observe significant improvements.

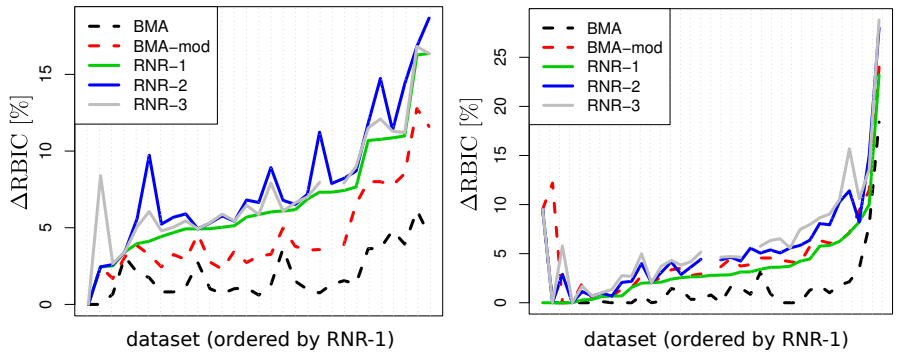

Fig. 3: Support improvement (in %) for optimization with a MRC (**left**) and SC (**right**) threshold. **RNR-l** depicts RNR runs with $l \in [1, 3]$, **BMA-mod** is a less conservative modification of the BMA. Data points missing because of prohibitive runtimes.

We suspect that, the unfavorable data locality inherent to phase II is the main reason for the moderate speed-up values.

We used the parallel implementation of RogueNaRok to optimize the support (for MRC and SC thresholds, $l = 1$) in 4 tree collections that are particularly large in number of taxa (*unpublished*, if not indicated otherwise): (i) 672 trees with 37,831 taxa, (ii) 211 trees with 55,593 [12], (iii) 200 trees with 77,215 taxa and (iv) 118 trees with 116,334 taxa. On these tree sets, our algorithm recovered between 4.6% and 7.9% of the maximum possible support and pruned between 1,312 and 8,684 taxa. For instance, the support of the strict consensus tree of the 55,593-taxon dataset was increased from RBIC = 7.7% to 12.4 by pruning 2,037 taxa. Execution times with 48 threads ranged between 11 and 130 hours, memory requirements between 30 and 40 GB. We expect runs for $l := 2$ to take between 7 and 50 days.

4.3 Qualitative Improvement

In this section, we evaluate to which degree various parameters for the RNR algorithm improve the support in a SC or MRC tree and compare the performance of our algorithm to that of the BMA. In general, comparison to the BMA is difficult, since its optimality criterion penalizes dropsets by the number of taxa that are pruned. We adapted BMA for obtaining an estimate to which degree a less conservative version (referred to as BMA-mod) is able to recover resolution. To this end, we changed its scoring scheme for a dropset d to $\frac{|b'|}{|d|}$, where $|b'|$ is the approximate increase of resolution and $|d|$ is the dropset size. For BMA-mod, the final dropsets as determined before termination, usually exhibit a strong detrimental effect on the support recovered so far. We account for that by explicitly calculating the optimality after each iteration of the algorithm (which significantly increases runtime) and stop when support can not be further improved.

Fig. 3 depicts the RBIC improvements obtained by the BMA, BMA-mod, and the RNR algorithm (with $l := 1$, $l := 2$ and $l := 3$). Overall, BMA-mod recovers significantly more support than than the default BMA. For a MRC threshold, RNR-1 performs consistently and significantly better than the BMA and BMA-mod. While RNR-1 still performs better than BMA, we have to set $l > 1$ to outperform BMA-mod. This is consistent with our previous observations [2], that is, BMA is more accurate when a SC threshold is used. Surprisingly, with a MRC threshold, RNR may yield less optimal results for larger maximum dropset sizes l. We suspect that complex rogue taxon scenarios are rare and that the algorithm starts to "over-prune", if l is chosen too large.

5 Conclusion And Outlook

We have introduced a graph-based formulation and an efficient algorithm for the rogue identification problem. We hope that, our "merger graph" concept will either give rise to a polynomial time algorithm or help to show that MISC is $\mathcal{NP}$-hard. Based on the graph formulation, we designed the RogueNaRok algorithm, which iteratively computes all minimal dropsets and determines how the optimality score changes when all possible minimal dropsets are pruned. Our algorithm is parametrized to only partially compute the graph of merger events between bipartitions. The parameter allows for trading potential result optimality for reduced execution times. The parameter also allows for re-using dropset information computed in previous iterations. For a maximum dropset size of $l := 1$, our implementation is up to 4,040 times faster than our previous algorithm, while yielding equivalent or identical result. We plan to make available a RogueNaRok-based web service, that will allow users to interactively explore the space of possible dropsets.

A straight-forward parallelization of the algorithm allows for identifying rogues in extremely large tree sets with respect to the number of taxa (up to 116,334 taxa). Further improving the parallel performance of RogueNaRok is a challenging engineering task that is beyond the scope of this study. Moreover, phase II of the algorithm could be accelerated, if we updated instead of recomputed connected components.

We also show that, RogueNaRok qualitatively outperforms a competing method. However, when using a SC threshold, more expensive computations are necessary than for MRC. For the MRC threshold, larger maximum dropset sizes sometimes yield a less optimal result. This behavior is not due to the RogueNaRok algorithm per se, but rather due to the greedy approximation strategy. A branch-and-bound approach may remedy this problem. On the other side, our observations on real datasets indicate that, the optimum can accurately be approximated using small dropset sizes.

Acknowledgment

We thank Stephen A. Smith for the datasets discussed in Section 4.3.

Bibliography

[1] Andre J Aberer. Advanced Methods for Phylogenetic Post-Analysis. Master's thesis, TU/LMU Munich, 2011.

[2] Andre J Aberer and Alexandros Stamatakis. A Simple and Accurate Method for Rogue Taxon Identification. *IEEE International Conference on Bioinformatics & Biomedicine*, 2011.

[3] Emery D Berger, Kathryn S McKinley, Robert D Blumofe, and Paul R Wilson. Hoard: a scalable memory allocator for multithreaded applications. *SIGPLAN Not.*, 35(11):117–128, November 2000.

[4] Casey W Dunn, Andreas Hejnol, David Q Matus, Kevin Pang, William E Browne, Stephen A Smith, Elaine Seaver, Greg W Rouse, Matthias Obst, Gregory D Edgecombe, Martin V Sø rensen, Steven H D Haddock, Andreas Schmidt-Rhaesa, Akiko Okusu, Reinhardt Mø bjerg Kristensen, Ward C Wheeler, Mark Q Martindale, and Gonzalo Giribet. Broad phylogenomic sampling improves resolution of the animal tree of life. *Nature*, 452(7188):745–749, April 2008.

[5] J Felsenstein, J Archie, W H E Day, W Maddinson, C Meacham, F J Rohlf, and D Swofford. The Newick tree format, 1986.

[6] Mark Holder and Paul O Lewis. Phylogeny estimation: traditional and Bayesian approaches. *Nat Rev Genet*, 4(4):275–284, April 2003.

[7] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[8] W P Maddison and D R Maddison. Mesquite: a modular system for evolutionary analysis, 2010.

[9] Nicholas Pattengale, Andre Aberer, Krister Swenson, Alexandros Stamatakis, and Bernard Moret. Uncovering Hidden Phylogenetic Consensus in Large Datasets. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, X(X):1–11, February 2011.

[10] Michael J Sanderson and H Bradley Shaffer. Troubleshooting molecular phylogenetic analyses. *Annual Review of Ecology and Systematics*, 33(1):49–72, 2002.

[11] Aaron B A Shafer and Jocelyn C Hall. Placing the mountain goat: a total evidence approach to testing alternative hypotheses. *Mol Phylogenet Evol*, 55(1):18–25, April 2010.

[12] S A Smith, J M Beaulieu, A Stamatakis, and M J Donoghue. Understanding angiosperm diversification using small and large phylogenetic trees. *American Journal of Botany*, 98(3):404, 2011.

[13] Erik A Sperling, Kevin J Peterson, and Davide Pisani. Phylogenetic-signal dissection of nuclear housekeeping genes supports the paraphyly of sponges and the monophyly of Eumetazoa. *Mol Biol Evol*, 26(10):2261–2274, 2009.

[14] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, November 2006.

[15] Robert C Thomson and H Bradley Shaffer. Rapid progress on the vertebrate tree of life. *BMC Biol*, 8:19, 2010.

[16] Robert C Thomson and H Bradley Shaffer. Sparse supermatrices for phylogenetic inference: taxonomy, alignment, rogue taxa, and the phylogeny of living turtles. *Syst Biol*, 59(1):42–58, January 2010.

[17] Thorley and Wilkinson. Testing the phylogenetic stability of early tetrapods. *J Theor Biol*, 200(3):343–344, 1999.

[18] M Wilkinson. Majority-rule reduced consensus trees and their use in bootstrapping. *Mol Biol Evol*, 13(3):437–444, 1996.

[19] Mark Wilkinson. Common Cladistic Information and its Consensus Representation: Reduced Adams and Reduced Cladistic Consensus Trees and Profiles. *Systematic Biology*, 43(3):343–368, 1994.

[20] Mark Wilkinson. More on Reduced Consensus Methods. *Systematic Biology*, 44(3):pp. 435–439, 1995.