Efficient Maximum Likelihood Tree Building Methods

Alexandros Stamatakis, Alexey M. Kozlov

Chapter 1.2 Efficient Maximum Likelihood Tree Building Methods

Alexandros Stamatakis
Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies,
Karlsruhe Institute of Technology, Institute for Theoretical Informatics
Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany
Alexandros.Stamatakis@h-its.org
https://orcid.org/0000-0003-0353-0691

Alexey M. Kozlov
Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies
Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany
Alexey.Kozlov@h-its.org
http://orcid.org/0000-0001-7394-2718

Abstract

The number of possible unrooted binary trees (phylogenies) increases super-exponentially with the number of taxa. To find the Maximum Likelihood (ML) tree one has to enumerate and evaluate all these trees. As we will see, this is computationally not feasible. Therefore, one predominantly deploys *ad hoc* tree search methods that strive to find a "good" ML tree in the hope that it will be close, either with respect to the likelihood score or the topological structure, to the globally optimal ML tree. In this chapter we provide an overview over the most popular and efficient ML tree search techniques.

How to cite: Alexandros Stamatakis and Alexey M. Kozlov (2020). Efficient Maximum Likelihood Tree Building Methods. In Scornavacca, C., Delsuc, F., and Galtier, N., editors, *Phylogenetics in the Genomic Era*, chapter No. 1.2, pp. 1.2:1–1.2:18. No commercial publisher | Authors open access book. The book is freely available at https://hal.inria.fr/PGE.

Funding This work was funded by the Klaus Tschira Foundation.

1 Introduction

The number of possible phylogenetic trees grows super-exponentially with the number of taxa. In many cases such a combinatorial explosion means that the optimization problem, that is, finding the ML tree, is what is called *NP-hard* in computer science.

In simple words, NP-hardness means that there does not exist any known algorithm for solving the problem requiring polynomial runtime as a function of the input size. In our case, the input size is the number of taxa and number of sites of the input, that is, the Multiple Sequence Alignment (MSA) of the taxa for which we desire to infer a tree. Throughout this chapter we will assume that the MSA is given.

Using the machinery of theoretical computer science, it has been formally proved that finding the optimal tree for character-based tree scoring criteria such as parsimony (Day, 1987) and ML (Roch, 2006) is NP-hard. In other words, we will need to calculate the ML score of every single possible tree to find the ML tree.

With the computer power available today, this might, at first glance not appear to be problematic. Assume, however, that we want to infer a ML tree on a MSA with 100 taxa which is, by current standards, only a medium-sized dataset with respect to the number of

Phylogenetics in the genomic era.
Editors: Celine Scornavacca, Frédéric Delsuc and Nicolas Galtier; chapter No. 1.2; pp. 1.2:1–1.2:18


taxa. For 100 taxa there exist roughly 1.7×10^{182} distinct unrooted phylogenies as calculated by our `TreeCounter` tool that we have found helpful for teaching purposes (available at `https://github.com/stamatak/TreeCounter`).

The exact number of possible unrooted binary trees for 100 taxa is:

17004588092934096228378478705035416073577250184104249002278358683636258088862833248513190100969641161111329025095469462826421330039198981168292392933990872222474946042895317077636718 75

Further, for the sake of simplicity, assume that calculating the ML score on one tree takes 1 second. To find the ML tree, one has to score all 1.7×10^{182} phylogenies. This requires an overall runtime of roughly 5×10^{174} years. In turn, this corresponds to only about the 3.8×10^{164}-fold age of our universe. It is important to note that, using powerful parallel supercomputers does not help to reduce this comparatively long waiting time as supercomputers reduce running times linearly (w.r.t., the number of processors they have) at best, but unfortunately, not super-exponentially. Assuming that we could use a rather large supercomputer with 10^{12} processors[1], we would still have to wait for the 3.8×10^{152}-fold age of the universe for the ML tree.

As we presumably do not want to wait for this long, we need to devise heuristic search strategies that navigate through this enormous space of phylogenies in an "intelligent" way and return a tree with a "good" score. As mentioned before, most of the heuristics currently used are *ad hoc* strategies. In other words, they do not offer any theoretical guarantees of how close or far away (regarding the ML score) the tree they return is from the globally optimal tree. This is also the reason why the sloppy terminology that we often observe in empirical evolutionary biology papers is potentially misleading. Papers often refer to "the ML tree". However, this is simply the best phylogeny found by the completely *ad hoc* heuristic search strategy.

Given the prolegomena, one might wonder what the computational complexity of Bayesian Inference of phylogenies might be, since it essentially also relies on repeated likelihood evaluations on trees (see Chapter 1.4 [Lartillot 2020a]). One would assume that their complexity must also somehow be affected by the vastness of tree space. If we simply look at Bayes' equation:

$$P(T|D) = \frac{P(T) \times P(D|T)}{P(D)} \tag{1}$$

where T is the tree, D the data, $P(T)$ is the prior probability and $P(D|T)$ the standard phylogenetic likelihood (as used for ML inference) score of the tree. The computational problem is hidden in $P(D)$ that represents the marginal probability of the data which cannot be computed exactly but needs to be approximated. An exact evaluation of this term would, again, as for ML, require calculating the likelihood scores of all possible topologies and, in the Bayesian setting, also for all possible remaining parameter values (e.g., branch lengths, rates of nucleotide substitution etc.). So calculating $P(D)$ exactly would require even longer waiting times than for ML above. As a consequence, one would rather not opt to calculate the posterior probability $P(T|D)$ exactly. As a work-around, the posterior probability is approximated via Markov-Chain Monte-Carlo (MCMC) methods. Unfortunately, these MCMC chains are only guaranteed to converge to the true posterior distribution if they are

[1] Evidently, no such supercomputer exists yet.

run for infinity. In practice, the lack of MCMC convergence can be assessed using so-called convergence analysis tools (e.g. Nylander et al., 2008). It can not be emphasized frequently enough that, these methods can not be used to demonstrate that the chains have converged. Based on the mathematical foundations of MCMC the chains will only converge to the true posterior probability distribution if executed eternally. Convergence analysis tools can therefore only detect lack of convergence. If there is no lack of convergence, then what they do indicate is that the MCMC process has reached an area of *apparent* convergence.

In the following sections we will introduce ML tree search algorithms in a top-down fashion. Initially, we discuss the commonalities of ML search strategies in Section 2. Subsequently we discuss the most common mechanisms for changing tree topologies in Section 2.2 and also cover some important implementation details (Section 2.3). In Section 3 we discuss how search strategies use these tree change moves to find "good" trees. We also discuss why we think that divide-and-conquer strategies for ML-based tree inference have not been successful to date (Section 3.1) and how terraces in tree space affect tree searches (Section 3.2).

In Section 4 we discuss the computation of support values using the standard phylogenetic bootstrap procedure and take a critical look at some recently proposed methods for obtaining approximate support values. We conclude in Section 5 with some notes of caution and recommendations on how to best infer a phylogeny from our point of view with a focus on selecting the best search strategy. For the sake of clarity, we deliberately omitted other important topics such as model selection or the selection of the most appropriate partitioning scheme.

2 Top-Level View of Search Algorithms

Initially, we will discuss the basic components that form part of almost every ML-based tree search algorithm. Most search strategies comprise the following two steps:

1. Construct an initial comprehensive tree that contains all taxa of the input MSA and compute its ML score
2. Start changing this initial comprehensive topology to find a tree with a better ML score

The most widely-used tools for ML-based inference (RAxML (Stamatakis, 2014), IQ-Tree (Nguyen et al., 2015), PHYML (Guindon et al., 2010), and GARLI (Zwickl, 2006)) implement this strategy. However, researchers have also experimented with divide and conquer approaches where the comprehensive tree is assembled by puzzling together independently optimized smaller subtrees. We discuss these approaches in more detail in Section 3.1.

2.1 Constructing Comprehensive Trees

There are at least three solutions to constructing a comprehensive starting tree: random topology, neighbor-joining (NJ) or its variants such as BioNJ (Gascuel, 1997), and parsimony. As predominantly done in Bayesian phylogenetic inference (since by definition a randomized stochastic sampling should start at a random point), one can simply construct a complete random starting tree. Alternatively, one might opt to infer a somewhat reasonable tree using a simpler tree building method such as neighbor-joining (NJ). A NJ starting tree typically has a better likelihood score than a complete random tree. On the other hand, just using the NJ tree, might drive the subsequent tree search into a local optimum. Because of the immense vastness of tree space that might exhibit a plethora of local optima, it is thus desirable to implement mechanisms (usually relying on some sort of randomization) that allow for navigating out of local optima (see Figure 8 for an example of a local optimum)

in some way. This can either be achieved by generating a distinct set of starting trees or via a randomization step in the tree search procedure. For now, we will focus on generating distinct starting trees. If our tool deploys complete random starting trees, obtaining a set of n distinct starting trees is straight-forward as we simply need to generate n comprehensive trees at random.

If we want to obtain a set of distinct starting trees with a better (i.e., non-random) initial likelihood score than a random tree, we can deploy the so-called randomized stepwise addition algorithm described below, using a simple criterion of our choice. Simple, in this context means, cheap-to-compute with respect to the computational cost of likelihood calculations. The parsimony or least-squares criteria are good examples of such simple criteria. Such reasonable yet simple criteria will generate starting trees that have a "good" initial likelihood score. Therefore, the subsequent tree search on the comprehensive tree (see Section 2.2) is likely to converge faster, hence reducing inference times. See Figure 1 for an example.

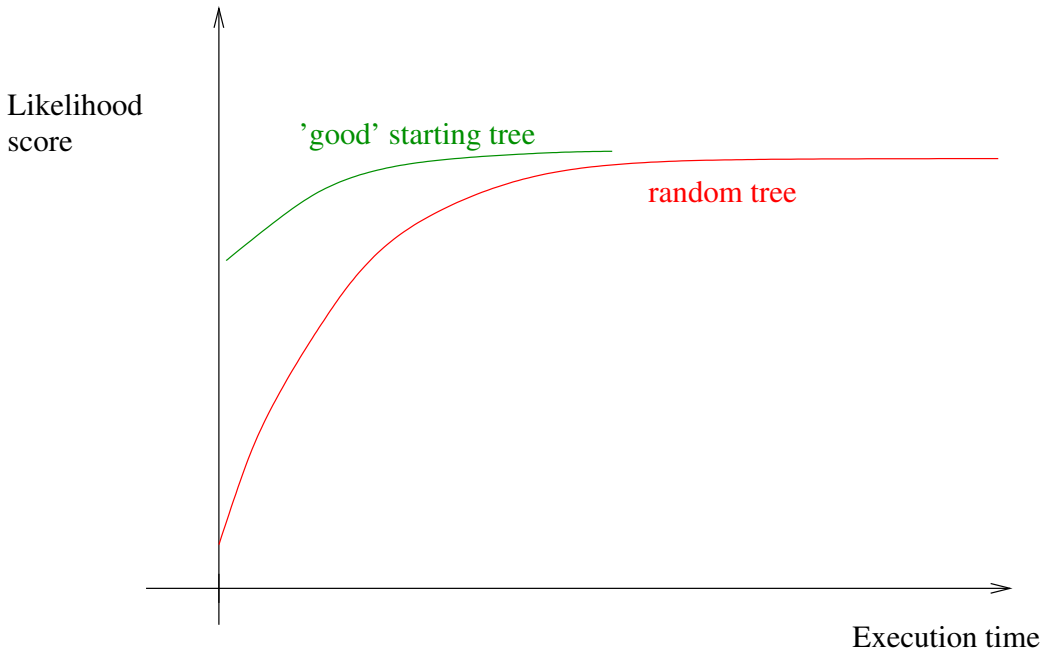

Figure 1 Schematic ML run time differences when initiating tree searches on random versus "good" comprehensive trees.

In the following, we will briefly outline the randomized stepwise addition order algorithm given a MSA with n taxa. The algorithm is also outlined in Figure 2.

1. Chose three taxa t_1, t_2, t_3 at random and use them to construct the only possible unrooted strictly binary three taxon tree
2. Chose the next taxon t_i to insert at random from the list of the remaining $n-3$ taxa
3. Insert t_i into every branch of the already constructed tree that has $i-1$ taxa. For each insertion of t_i into a branch calculate and store the score using, for instance, parsimony. Then, remove t_i from the current branch again.
4. Once, we have computed insertion scores for all branches, finally insert t_i into the branch that yielded the best insertion score.
5. Continue adding taxon after taxon to the tree as above until no taxa are left to insert.

In general, applying this procedure several times using distinct randomized taxon addition orders (e.g., inserting in this order t_1, t_2, t_3, t_4, t_5 versus inserting in the following order t_3, t_1, t_5, t_2, t_4) we will obtain a set of topologically distinct comprehensive initial trees. However, if the signal in the data is very strong (e.g., large concatenated supermatrices as in Misof et al. (2014) it might well be, and we have observed this while analyzing empirical datasets, that several distinct addition orders do yield the same tree. Therefore, one should first check, for instance, by computing the Robinson-Foulds distances (Robinson and Foulds, 1981) between all pairs of starting trees (e.g., using an appropriate script or `-rfdist` command of RAxML-NG), how many distinct trees the inferred starting tree set does contain, prior to launching computationally intensive ML searches on these trees.

One might wonder, why we typically do not use likelihood as a criterion for this randomized stepwise addition procedure. This is simply again due to the computational cost of ML. Note that, evaluating likelihoods on trees takes between 85%-95% of overall runtime in all likelihood-based tree inference tools (this also holds for Bayesian inference). In addition, using likelihood in this step does not yield substantially better trees than using parsimony (Morrison, 2007). Moreover, one will typically observe larger likelihood improvements by applying topological changes to the comprehensive tree (see Section 2.2 below). Thus, using parsimony or NJ/BioNJ represents a classic engineering trade-off between the quality (likelihood score) of the initial tree and the time required to construct it. For the sake of completeness, it is worth mentioning that some older tools, namely fastDNAml (Olsen et al., 1994) and PAUP* (Swofford, 2001), implemented a randomized stepwise addition procedure using Maximum Likelihood as an option.

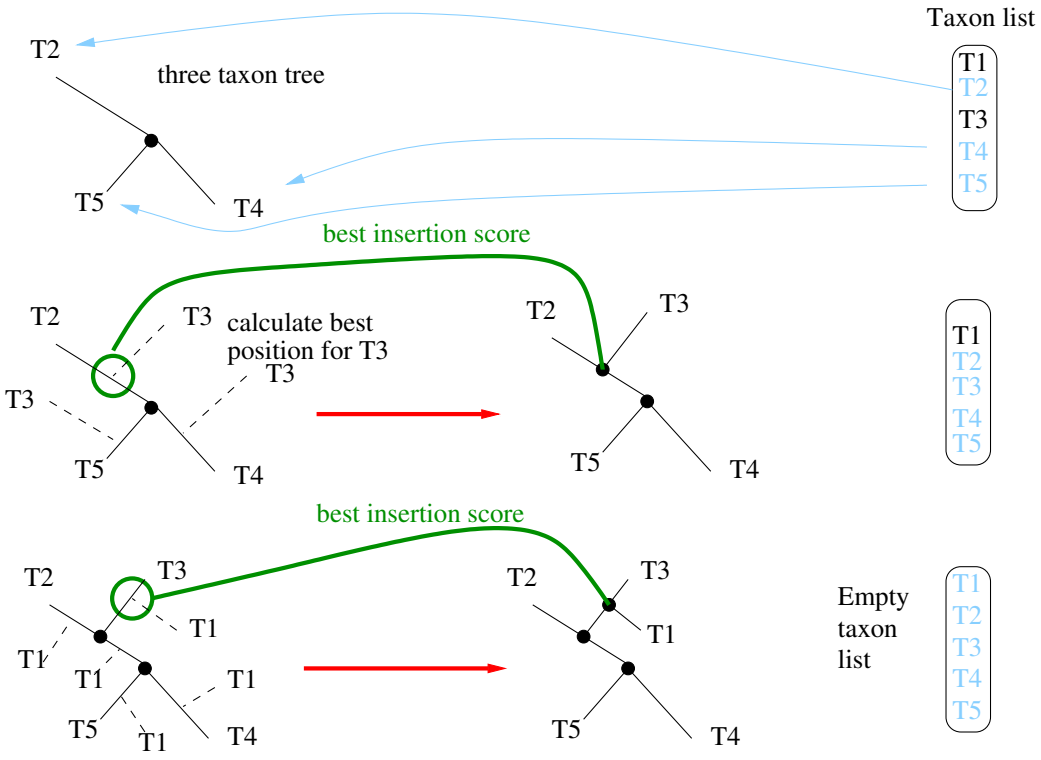

Figure 2 Example of randomized stepwise addition algorithm for constructing an initial tree comprising 5 taxa.

To conclude, using parsimony-based randomized stepwise addition order starting trees

(as in RAxML or IQ-Tree) might potentially bias the search towards specific parts of the tree space and particular local optima. Therefore, search strategies relying on comprehensive starting trees and subsequent greedy hill climbing can benefit from using different starting tree types (e.g., *both* random *and* parsimony) to more thoroughly explore tree space. In general, this needs to be assessed on a case by case basis, depending on the data at hand. The way we imagine the search space is visualized in Figure 3.

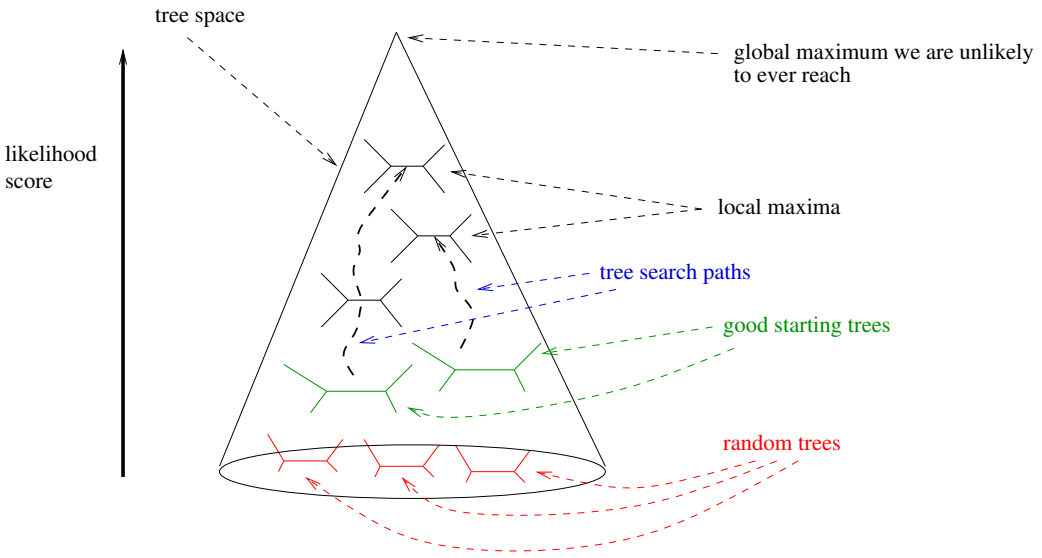

Figure 3 Our way of imagining tree search space, including random starting trees, "good" starting trees, and tree search paths that take us closer to the desired global maximum, that is *the* ML tree.

As already mentioned, most Bayesian inference programs typically start from a random tree. However, some implementations (MrBayes (Ronquist et al., 2012), ExaBayes (Aberer et al., 2014)) do offer the option to also initiate the MCMC procedure on a randomized stepwise addition order parsimony tree.

2.2 Changing Topologies - Searches on Comprehensive Trees

Now that we know how to compute a comprehensive starting tree, we can consider the basic techniques for changing that tree in order to further improve the likelihood.

We will first only consider the widely-used standard tree moves that allow us to change the tree to varying degrees. In other words, we will consider bold (change the tree topology substantially) versus conservative (slightly change the tree) topological alteration mechanisms. We will discuss how to use these standard mechanisms to build a tree search strategy in Section 3 including several examples. When discussing moves, we always refer to a given tree as the current tree. This can, for instance, be the best tree we have found so far and that is stored in memory. Via a tree move we then attempt to construct a new tree by changing the current tree in the hope that this new tree will have a better likelihood than the current tree.

The three most widely used fundamental tree moves (also referred to as topological alteration mechanisms) are the following:

1. NNI: Nearest Neighbour Interchange (see Figure 4)
2. SPR: Subtree Pruning and Re-grafting (see Figure 5)

3. TBR: Tree Bisection and Reconnection (see Figure 6)

The most simple as well as most conservative move is the NNI move. To apply a NNI move we first need to select (how we do this selection is a matter of designing the search strategy) an inner branch of the tree that defines a blue (B), a yellow (Y), a red (R), and a green (G) subtree. In our example in Figure 4 subtrees B and R are located on one side of the branch and Y and G on the other side. To generate alternative trees with NNI we can now flip our colored subtrees over the inner branch. Thereby, from the tree containing this inner branch, we can construct two alternative, not substantially different (in terms of topological distance to the initial tree; for bolder moves see SPR and TBR explained below), tree topologies and evaluate their likelihood scores.

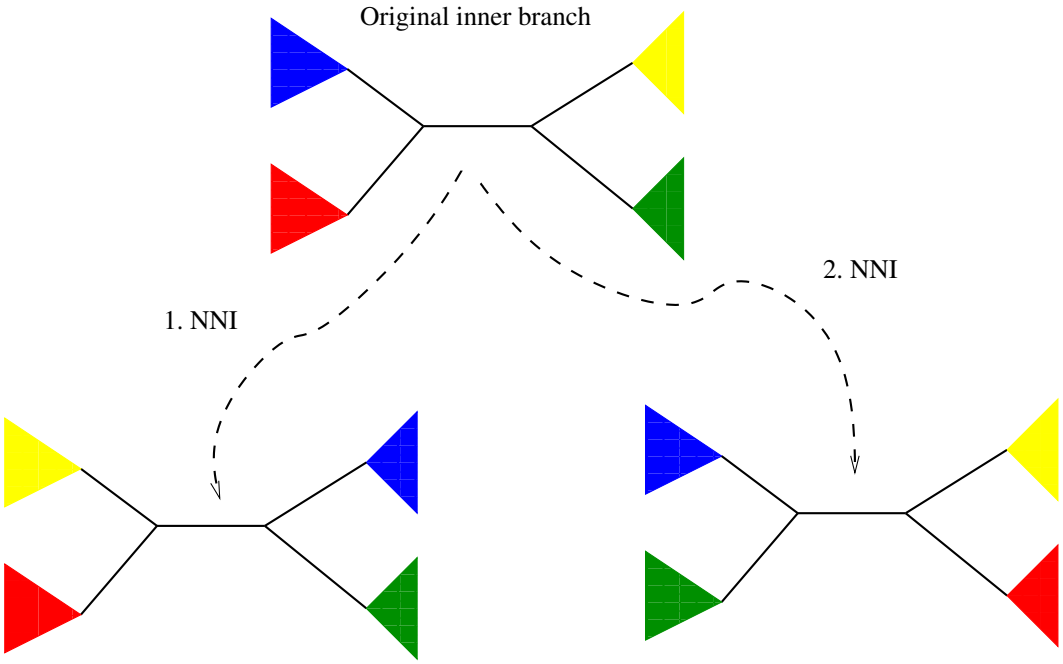

Figure 4 Outline of the two possible NNI moves as executed on an inner branch of a phylogenetic tree defining red, blue, green, and yellow subtrees.

A more bold move than the NNI is the SPR move. To carry out a SPR move, we first select the root of a subtree in the comprehensive tree. Again, how and in which order we select such a root depends on the actual tree search strategy. In RAxML, for instance, we conduct a depth first traversal of the tree and apply SPR moves to *all* subtrees we encounter. Once we have selected a subtree root, we initially prune the subtree (called *candidate subtree*) by removing its subtree root from the branch to which it is attached to. We call the original branch in the current tree where the subtree was pruned the *pruning branch*. Then we can start inserting, calculating the likelihood, and removing again our candidate subtree into the neighboring branches of the pruning branch. We can define the size of this neighborhood by the so-called rearrangement radius (see Figure 5). The rearrangement radius allows us to determine up to how many nodes away from the pruning branch we desire to re-insert our candidate subtree. If we set the rearrangement radius to 1, we will only execute conservative SPR moves, whereas if we set the rearrangement radius to the total number of inner nodes in the tree (remember that this is $n - 2$ for an unrooted binary tree with n taxa), we perform bold moves and explore a larger portion of the colossal tree space. How to set this

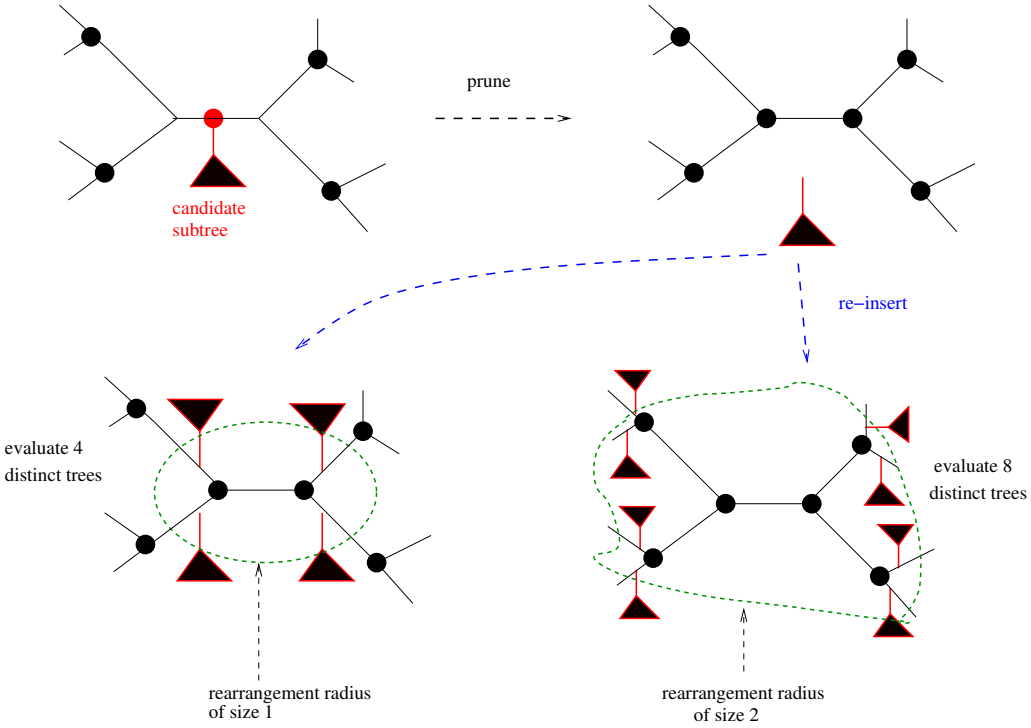

Figure 5 Outline of the Subtree Pruning and Re-Grafting Procedure. First, a candidate subtree is selected. Then, it is pruned from the current tree. Subsequently, one can re-insert it at all branches that are one node away from the pruning branch (rearrangement radius of 1), all branches that are two nodes away from the pruning branch (rearrangement radius of 2), etc.

rearrangement radius, how to adapt it to the MSA at hand, and how to potentially change it over the course of a tree search is again subject to search strategy development. In RAxML, for instance, the default behavior is to automatically determine a "good" rearrangement radius using the following strategy (implementation details omitted): choose the smallest SPR radius that yields the largest likelihood improvement on the starting tree. In most other tools, the rearrangement radius is set to a fixed default value, but can be changed by the user via respective command line flags.

The most drastic tree move is the TBR move (see Figure 6). A TBR move is drastic because it can induce large topological changes on the tree. As a consequence, a TBR move can also either substantially increase or decrease the likelihood of a tree. To execute a TBR move one initially selects a branch to *bisect* the current tree to obtain two unconnected subtrees t_1 and t_2. Then, one generates alternative tree topologies by reconnecting the two subtrees. This is accomplished by connecting all pairs of branches in the two unconnected subtrees via a new branch. Thereby, one obtains $n \times m$ alternative tree topologies, where n is the number of branches in t_1 and m the number of branches in t_2. As for SPR moves, one can limit the range of these reconnection operations to a neighborhood around the respective positions from which the original branch was removed.

2.3 Implementation Details

The three fundamental tree moves appear to be relatively straight-forward. To maximize computational efficiency, that is, to minimize the amount of phylogenetic likelihood calcu-

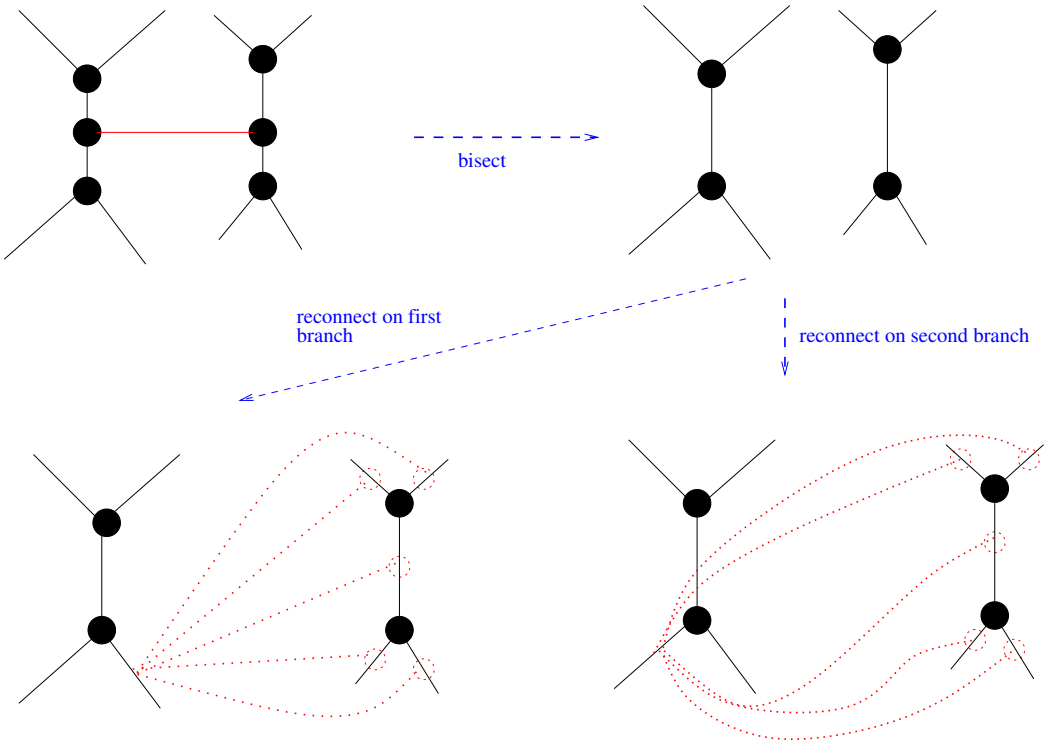

Figure 6 Outline of the tree bisection and reconnection move. The tree is initially bisected by removing the red branch in the top left corner of the figure to obtain two disjoint subtrees t_1 and t_2. Then, we can start reconnecting them by visiting each branch of the left subtree and connecting it with all branches of the right subtree as shown for two out of the five branches of the left subtree in the bottom of the figure.

lations, a plethora of shortcuts and heuristics are applied in practice. In principle, after each tree move, for instance, after applying a NNI move, one would need to re-optimize *all* branch length values and *all* remaining model parameters (e.g. GTR rates, α shape parameter of the Γ distribution to model rate heterogeneity) to obtain *the* ML score of the tree generated via the move. As such global optimization operations on phylogenies are highly compute-intensive because they require repeatedly traversing and re-computing the likelihood on the entire tree, all common ML based tree inference programs use shortcuts. In other words, they only compute an approximate likelihood score for a tree generated via a tree move and not *the* ML score.

Typically there are three types of shortcuts that are used by ML software developers.

2.3.1 Avoiding model parameter optimization

To circumvent the high computational cost of re-estimating the ML model parameters (GTR rates, α parameter) with the exception of the branch lengths, one relies on the following empirical observation: As long as the tree we are applying our moves to is reasonable (i.e., non-random) the model parameter estimates will not change substantially. Thus, it is sufficient to only re-estimate them periodically after having applied a relatively large number of tree moves and potential changes to the tree topology.

2.3.2 Avoiding global branch length optimization

To avoid re-estimating all branch lengths of the new tree after a move, program developers rely on the following intuition: The branch lengths that are in the neighborhood where the tree was changed should be affected most by the move. Hence, only those supposedly most affected branch lengths are typically re-estimated. Consider, for instance, a NNI move. Here, one assumes that the 5 branches shown in black in Figure 4 are affected most by the NNI move. Thus, one would only re-estimate these 5 branch lengths after applying a NNI. One could even decide to only re-estimate the center branch. What works best is a matter of numerous trial-and-error experiments on benchmark datasets during program development. For SPR moves one can apply the so-called lazy SPR move technique that was introduced in RAxML. Here, only the three branch lengths adjacent to the subtree insertion position are re-estimated (see Figure 7). In GARLI there is a more elaborate method for conducting lazy SPR moves. GARLI tries to dynamically determine the number of branches that need to be re-estimated after a move. In other words, it optimizes branch lengths at an increasing distance from the subtree insertion position until they do not change significantly any more.

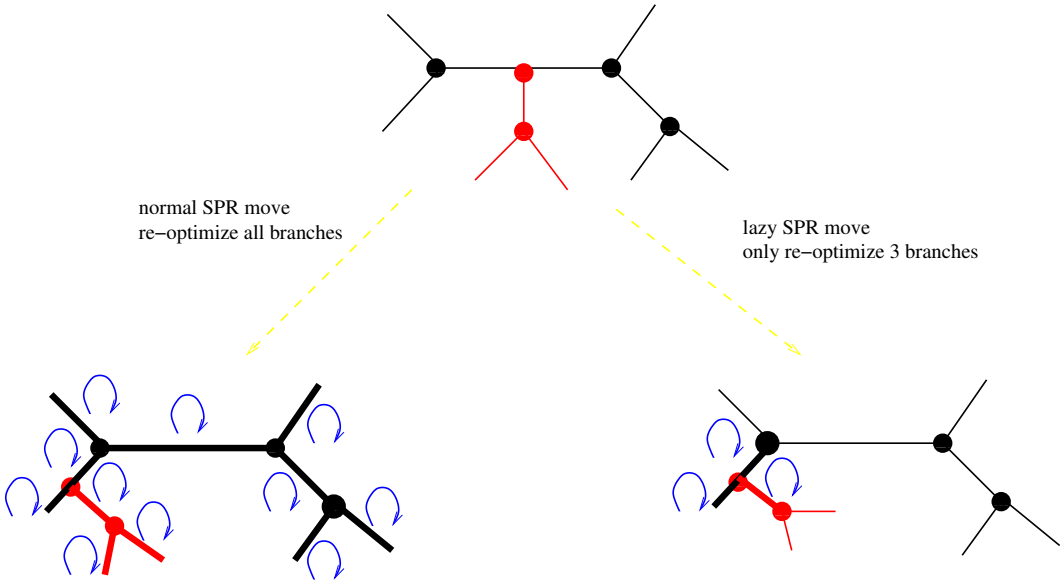

Figure 7 Outline of a standard versus a lazy SPR move on the subtree with the red branches. In the left hand bottom corner we conduct a standard SPR move, that is, we re-optimize *all* branch lengths of the tree (indicated by thick lines). In the right hand bottom corner we conduct a lazy SPR move, that is, we only re-optimize the three branches (shown by thick lines) that are adjacent to the insertion position of the subtree.

2.3.3 Locality of tree move applications

To avoid jumping back and forth between different distant regions of the tree while applying topological moves, most ML search algorithms apply moves systematically to the tree in a pre-defined order (e.g., a depth first traversal of the tree). This improves computational efficiency as the moves (and hence the costly updates of Conditional Probability Vectors) are always taking place in only one region of the tree, while the rest of the tree remains unaltered. Then, one moves on to a neighboring region (e.g., a neighboring subtree which one tries to rearrange with SPR moves).

3 Search Strategies

Given our set of fundamental tree moves, we can now design tree search strategies. Most popular search algorithms repeatedly apply one or several of these moves to the current tree until there is no move that yields a tree with a better ML score. Once, no move can be applied that further improves the ML score, the algorithm converges and returns the best-scoring ML tree it was able to find.

A simple search strategy as implemented in the original version of PHYML (Guindon and Gascuel, 2003)

1. Build a NJ starting tree.
2. Repeatedly apply NNI moves to all inner branches of the tree.
3. Terminate if for none of the inner branches there is a NNI move that can further improve the ML score of the tree.

The key pitfall of such a simple search strategy is that it is highly likely to get stuck early in a local optimum as outlined in Figure 8.

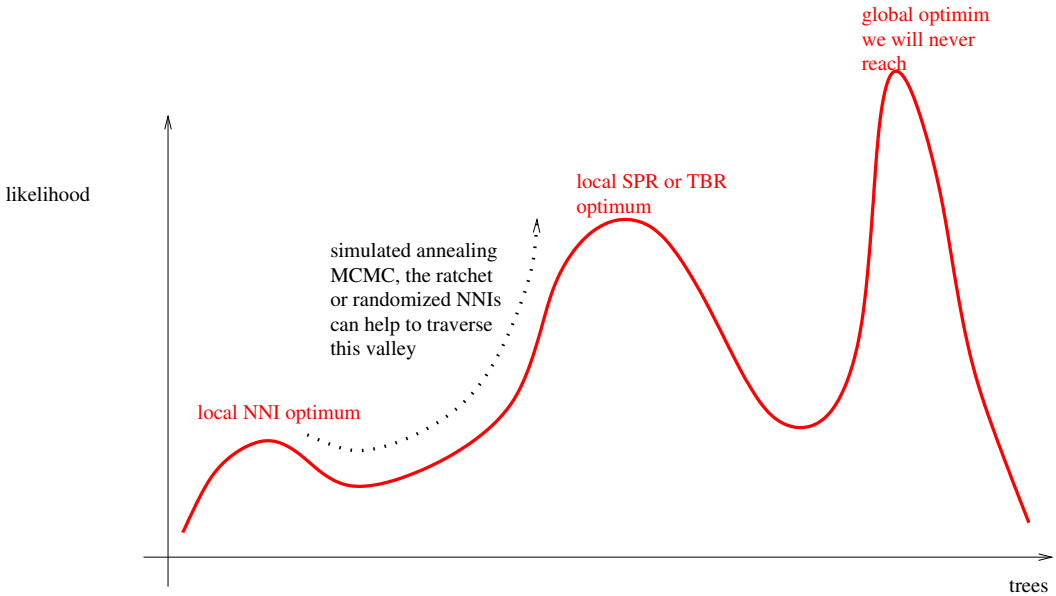

Figure 8 Example of how ML searches can get stuck in local optima. The first local optimum on the right could be a local NNI-optimum, that is, we are not able to navigate out of this optimum by applying NNI moves. One could move from this optimum to the better local optimum in the middle of the graph via SPR or TBR moves or by applying a stochastic search (e.g., simulated annealing or MCMC in the Bayesian setting. To the right we see the global optimum which we are unlikely to ever reach.

One way to alleviate this is to use more radical moves such as SPR or TBR that allow to more easily move away from such a local optimum again. RAxML and GARLI mainly rely on lazy SPR moves, while PHYML version 3 also introduced SPR moves (Guindon et al., 2010).

In addition, one can conduct multiple tree searches on several distinct randomized addition order parsimony starting trees (e.g., RAxML [Stamatakis 2014] or IQ-Tree (Nguyen et al., 2015)). While these search strategies are still highly likely to be stuck in local optima when

they converge, those local optima will have higher ML scores than the local optima the above simple algorithm will become stuck in.

As IQ-Tree mainly relies on NNI moves and is thus more prone to getting stuck in local optima, apart from using distinct parsimony starting trees, it deploys an additional technique. Once it is stuck in a so-called NNI optimum, it applies a couple of completely random NNI moves (without taking the likelihood score of these moves into account). This perturbs the tree sufficiently to move out of the local optimum. After this perturbation, a new round of NNI moves for improving the ML score is applied which is highly likely to end up in a distinct, potentially better, local optimum.

An alternative approach for navigating out of local optima by means of random perturbations is the ratchet method that has been applied to parsimony (Nixon, 1999) and ML searches (Vos, 2003). Here, the idea is to first randomly change the weights of the MSA sites, then apply a couple of NNI or SPR moves to this perturbed MSA, and subsequently restore the original MSA. Thereafter, one re-applies the search strategy to the new tree generated by the perturbed MSA.

GARLI (Zwickl, 2006) deploys a rather different approach to escaping local optima. It uses a so-called genetic algorithm. Instead of working on a single tree, GARLI conducts searches on a set (population) of trees. Periodically, information (e.g., subtrees) is exchanged among the trees in the population to improve the quality (ML scores) of the overall tree population.

Yet another option for escaping local optima is to deploy the simulated annealing search technique. Simulated annealing allows for carrying out backward steps, that is, it will occasionally conduct tree moves to trees with lower ML scores. This might allow to navigate out of local maxima in parsimony (Barker, 2004) and ML-based (Stamatakis, 2005) tree searches.

It is worth noting that simulated annealing is very similar to Bayesian MCMC sampling, as MCMC chains also occasionally accept so-called downhill steps to sample trees with a lower posterior probability.

Finally, there also exists the issue of what we have termed "rough likelihood surface". Such a rough likelihood surface typically emerges when analyzing datasets with comparatively few sites and a large number of taxa (e.g., a single-gene 16S RNA alignment comprising thousand taxa, or more). Typically, the search space will exhibit a large number of local optima that (i) can not be distinguished from each other using the standard likelihood-based significance tests as implemented, for instance, in CONSEL (Shimodaira and Hasegawa, 2001) and (ii) that exhibit large pairwise topological distances exceeding 20% or even 30% on average. Stamatakis (2011) gave an example of such a rough likelihood surface using an empirical single-gene dataset. In general, the key challenge with such datasets is that 100 distinct ML searches are likely to yield 100 topologically substantially different, but statistically indistinguishable trees.

3.1 Divide-and-Conquer Approaches

Thus far, we have briefly discussed how the most popular ML tree search strategies work. An alternative approach to designing phylogenetic search strategies is to deploy a divide-and-conquer strategy. Here, the idea is to initially divide the input sequences into disjoint or partially overlapping sets of closely related sequences (e.g., simply by clustering sequences or using a parsimony tree), then infer individual trees on those subsets, and finally merge these subtrees into one large comprehensive tree. The merging step can also be interpreted as a supertree reconstruction problem, provided that the sequence subsets overlap.

In general, all attempts to devise efficient divide-and-conquer tree search algorithms for ML have not been particularly successful with respect to speed and/or accuracy improvements over the aforementioned search methods that operate on a comprehensive tree. It turns out that all divide-and-conquer methods that have been devised so far, require an additional global optimization of the tree topology, the branch lengths, and the model parameters on the comprehensive tree to compete (with respect to accuracy) with the standard approaches described in Section 3. This requirement for additional global optimizations on a comprehensive tree also has a negative impact on the potential speed savings a divide-and-conquer approach could have.

While this is not properly understood yet, we suspect that global optimization on the comprehensive tree *is* required, because the information on branch lengths and model parameters that is propagated from an individual subtree has an important influence on the tree topology and branch lengths in the remaining subtrees. In other words, the subproblems (subtrees) we are attempting to solve do not appear to be sufficiently independent from each other to allow for applying a divide-and conquer approach. Some examples for divide-and-conquer or similar approaches with rather disappointing results have been shown by Roshan et al. (2004), Le Vinh et al. (2005) and Izquierdo-Carrasco et al. (2011).

A somewhat related approach is the quartet puzzling idea as implemented in Tree-Puzzle (Schmidt et al., 2002). It initially builds quartet trees (trees with 4 taxa) from the MSA and then puzzles them together into a comprehensive tree.

3.2 Terraces in Tree Space

A recently discovered phenomenon affecting likelihood-based phylogenetic inference (ML and BI) are terraces in tree space (Sanderson et al., 2011). A terrace is a, potentially large set, of topologically distinct tree topologies with *exactly* the same analytical likelihood score. Note that, numerical likelihood values might differ slightly because of roundoff error propagation.

Under likelihood, terraces may emerge for partitioned phylogenomic alignments when using unlinked branch length estimates. Branches are said to be unlinked, when we estimate a completely independent set of branch lengths for each partition (e.g., each gene) of the phylogenomic MSA (also known as supermatrix in this context). Terraces only emerge when branch lengths are unlinked, because the overall likelihood score of the entire concatenated MSA is the sum of the independently *optimized* per-partition likelihoods. In contrast to this, when branch lengths are not unlinked (i.e., scaled or joint branch length estimates), they can not be optimized independently for each partition separately. In other words, the partitions are somehow connected to each other via the shared (potentially scaled) branch length values which prevents the emergence of terraces.

If an a MSA (under an unlinked branch model) and respective partitioning scheme contains one or several terraces depends on the missing data pattern (Sanderson et al., 2011). Therefore, in the context of designing tree searches, one should avoid conducting tree moves that will just take the search to another tree located on the same terrace, or at least omit such redundant computations. Tree moves omitting unnecessary likelihood computations in a prototype RAxML implementation were pioneered by Stamatakis and Alachiotis (2010), essentially through implicit recognition of terraces, even before they were mathematically characterized in 2011. Later on, Chernomor presented work on omitting redundant computations for standard tree move operations using more elegant data structures than we did as well as a production-level implementation in IQ-Tree (Chernomor et al., 2015, 2016).

While the techniques implemented in RAxML and IQ-Tree allow for avoiding redundant

likelihood calculations on a terrace, they do not provide explicit mechanisms to move away from the terrace in tree space.

Terraces are not only relevant with respect to reducing the computational cost of ML searches. Their presence can also mislead downstream analyses.

For instance, the presence of terraces can severely bias bootstrap and Bayesian support values (Sanderson et al., 2014). Moreover, it is currently unknown how many published phylogenetic trees actually do reside on a terrace. The first study devoted to this topic (Dobrin et al., 2018) analyzed a collection of 26 large empirical phylogenomic datasets and showed that terraces are present in "nearly all datasets" and that "terraces found during bootstrap resampling reduced overall support".

One reason for this is that standard phylogenetic inference tools do not routinely assess if the final tree they generated resided on a terrace or not. However, there now exists a highly efficient C++ library (Biczok et al., 2018) for this purpose, that has already been integrated into RAxML-NG.

4 Computing Support Values

An important aspect when conducting empirical phylogenetic studies is the inference of support values on trees. The standard method is the non-parametric bootstrap (BS) as proposed by Felsenstein (1985) (but also see an interesting very recent modification of the phylogenetic bootstrap by Lemoine et al., 2018) The idea is to re-sample sites from the original MSA at random with replacement to assemble a set of 100 or more slightly perturbed BS replicate MSAs. One then applies the same algorithm as used for inferring the best-known ML tree on the original —unperturbed— MSA to each of those BS replicate MSAs. This yields a set of 100 (or more) BS trees that can subsequently be used to, either build a consensus tree, or map branch support values to the best-known ML tree inferred on the original MSA. Finally, a more elaborate approach to mapping BS support values onto a given tree, that also takes the size of the respective tree space into account, has recently been presented (Lemoine et al., 2018).

How many BS replicates are required is hard to determine, but it seems to heavily depend on the data at hand. For a criterion to determine the number of BS replicates, see Pattengale et al. (2010).

Evidently, the bootstrap procedure is computationally extremely expensive. Thus, there have been several attempts to devise faster and hence more approximate methods for inferring support values on phylogenies. It is important to note that, they represent "approximations of an approximation" as finding the optimal ML tree is NP-hard and the strategies presented in Section 3 already represent *ad hoc* heuristics. As such, using these fast methods for inferring support values can always, and perhaps rightfully so, be criticized by reviewers as being too approximate. From our point of view, despite having designed an approximate method ourselves, they do not capture that well, the search complexity of the problem that is associated to the vast tree search space, and the mere fact that we are already using heuristics. Thus, from our personal point of view, it is always best to conduct BS replicate searches using the standard search strategies with the standard bootstrap procedure, if the dataset size and the available computational resources allow for this.

In RAxML we introduced the so-called rapid bootstrap (Stamatakis et al., 2008) that essentially relies on a more approximate, less thorough version of the standard RAxML search algorithm, that can more easily be trapped in local maxima.

The so-called ultrafast bootstrap (Minh et al., 2013) implemented in IQ-Tree relies on an

approximate sampling of emulated BS replicates during the search on the original MSA. As such, it depends heavily on whether the regions of tree space that are explored by the search on the original MSA also form part of the BS replicate tree search space. In other words, one needs those two regions (original MSA and BS replicate MSA region) of the tree search space to overlap sufficiently in order for the approximation to be accurate. This is, however, not guaranteed a priori.

Finally, the approximate Likelihood Ratio Test (aLRT, Anisimova and Gascuel, 2006) takes a given best-known ML tree and conducts a statistical test on each inner branch of the tree by computing likelihood scores for the three possible NNI configurations (see Figure 4) around this branch and subsequently using them for the test statistic. The aLRT can be criticized for exclusively relying on the very local NNI-based likelihoods for computing the test statistics.

We believe that all of the above approximations for obtaining support values are useful. We nonetheless wish to emphasize that they remain approximations of an approximation and are thus prone to criticism.

5 Conclusion

In this book chapter we have presented the basic components, tree alteration operations, and flavors of commonly used tree search strategies under ML. Moreover, we discussed the standard phylogenetic BS procedure for inferring support values on trees as well as some faster and more approximate methods for this task. We also briefly reviewed the time complexity for finding *the* ML tree and explained the intuition behind the concept of NP-hardness. Finally, we also briefly outlined the additional problems that arise when the dataset to be analyzed contains so-called terraces in its tree search space.

Practitioners should keep in mind that all tree search tools implement *ad hoc* heuristic search strategies that have been developed and tested using some simulated and some empirical benchmark test datasets. It is thus likely that they will fail, that is, perform sub-optimally on other datasets under distinct or difficult settings. Also, the search strategies presented here do not have any performance guarantees, that is, how far away from the global maximum the trees they infer are.

The best approach is to conduct searches using several ML tree inference tools (e.g. Chapter 1.3 [Kozlov and Stamatakis 2020]) as well as tools for Bayesian phylogenetic inference (e.g. Chapter 1.5 [Lartillot 2020b]) and subsequently compare the results. This also helps to minimize the impact of potential programming errors (Darriba et al., 2018) as tree inference software has become more complex and supports substantially more, as well as more complex models than 10 - 15 years ago.

Acknowledgements

The authors wish to thank the following former students of our 2018 summer school on computational molecular evolution for useful comments on the initial draft of this book chapter: Paschalis Natsidis and Alexandros Vasilikopoulos.

References

Aberer, A. J., Kobert, K., and Stamatakis, A. (2014). Exabayes: Massively parallel bayesian tree inference for the whole-genome era. *Molecular biology and evolution*, 31(10):2553–2556.

Anisimova, M. and Gascuel, O. (2006). Approximate likelihood-ratio test for branches: a fast, accurate, and powerful alternative. *Systematic biology*, 55(4):539–552.

Barker, D. (2004). Lvb: parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics (Oxford, England)*, 20(2):274–275.

Biczok, R., Bozsoky, P., Eisenmann, P., Ernst, J., Ribizel, T., Scholz, F., Trefzer, A., Weber, F., Hamann, M., and Stamatakis, A. (2018). Two c++ libraries for counting trees on a phylogenetic terrace. *Bioinformatics*.

Chernomor, O., Minh, B. Q., and von Haeseler, A. (2015). Consequences of common topological rearrangements for partition trees in phylogenomic inference. *Journal of Computational Biology*, 22(12):1129–1142.

Chernomor, O., von Haeseler, A., and Minh, B. Q. (2016). Terrace aware data structure for phylogenomic inference from supermatrices. *Systematic biology*, 65(6):997–1008.

Darriba, D., Flouri, T., and Stamatakis, A. (2018). The state of software for evolutionary biology. *Molecular biology and evolution*, 35(5):1037–1046.

Day, W. H. (1987). Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467.

Dobrin, B. H., Zwickl, D. J., and Sanderson, M. J. (2018). The prevalence of terraced treescapes in analyses of phylogenetic data sets. *BMC evolutionary biology*, 18(1):46.

Felsenstein, J. (1985). Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4):783–791.

Gascuel, O. (1997). Bionj: an improved version of the nj algorithm based on a simple model of sequence data. *Molecular biology and evolution*, 14(7):685–695.

Guindon, S., Dufayard, J.-F., Lefort, V., Anisimova, M., Hordijk, W., and Gascuel, O. (2010). New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0. *Systematic biology*, 59(3):307–321.

Guindon, S. and Gascuel, O. (2003). A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic biology*, 52(5):696–704.

Izquierdo-Carrasco, F., Smith, S. A., and Stamatakis, A. (2011). Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees. *BMC bioinformatics*, 12(1):470.

Kozlov, A. M. and Stamatakis, A. (2020). Using raxml-ng in practice. In Scornavacca, C., Delsuc, F., and Galtier, N., editors, *Phylogenetics in the Genomic Era*, chapter 1.3, pages 1.3:1–1.3:25. No commercial publisher | Authors open access book.

Lartillot, N. (2020a). The bayesian approach to molecular phylogeny. In Scornavacca, C., Delsuc, F., and Galtier, N., editors, *Phylogenetics in the Genomic Era*, chapter 1.4, pages 1.4:1–1.4:17. No commercial publisher | Authors open access book.

Lartillot, N. (2020b). Phylobayes: Bayesian phylogenetics using site-heterogeneous models. In Scornavacca, C., Delsuc, F., and Galtier, N., editors, *Phylogenetics in the Genomic Era*, chapter 1.5, pages 1.5:1–1.5:16. No commercial publisher | Authors open access book.

Le Vinh, S., Schmidt, H. A., and von Haeseler, A. (2005). Phynav: A novel approach to reconstruct large phylogenies. In *Classification—the Ubiquitous Challenge*, pages 386–393. Springer.

Lemoine, F., Entfellner, J.-B. D., Wilkinson, E., Correia, D., Felipe, M. D., Oliveira, T., and Gascuel, O. (2018). Renewing felsenstein's phylogenetic bootstrap in the era of big data. *Nature*, 556(7702):452.

Minh, B. Q., Nguyen, M. A. T., and von Haeseler, A. (2013). Ultrafast approximation for phylogenetic bootstrap. *Molecular biology and evolution*, 30(5):1188–1195.

Misof, B., Liu, S., Meusemann, K., Peters, R. S., Donath, A., Mayer, C., Frandsen, P. B., Ware, J., Flouri, T., Beutel, R. G., et al. (2014). Phylogenomics resolves the timing and pattern of insect evolution. *Science*, 346(6210):763–767.

Morrison, D. A. (2007). Increasing the Efficiency of Searches for the Maximum Likelihood Tree in a Phylogenetic Analysis of up to 150 Nucleotide Sequences. *Systematic Biology*, 56(6):988–1010.

Nguyen, L.-T., Schmidt, H. A., von Haeseler, A., and Minh, B. Q. (2015). Iq-tree: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular biology and evolution*, 32(1):268–274.

Nixon, K. C. (1999). The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15(4):407–414.

Nylander, J. A., Wilgenbusch, J. C., Warren, D. L., and Swofford, D. L. (2008). Awty (are we there yet?): a system for graphical exploration of mcmc convergence in bayesian phylogenetics. *Bioinformatics*, 24(4):581–583.

Olsen, G. J., Matsuda, H., Hagstrom, R., and Overbeek, R. (1994). fastdnaml: a tool for construction of phylogenetic trees of dna sequences using maximum likelihood. *Bioinformatics*, 10(1):41–48.

Pattengale, N. D., Alipour, M., Bininda-Emonds, O. R., Moret, B. M., and Stamatakis, A. (2010). How many bootstrap replicates are necessary? *Journal of computational biology*, 17(3):337–354.

Robinson, D. F. and Foulds, L. R. (1981). Comparison of phylogenetic trees. *Mathematical biosciences*, 53(1-2):131–147.

Roch, S. (2006). A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(1):92.

Ronquist, F., Teslenko, M., van der Mark, P., Ayres, D. L., Darling, A., Höhna, S., Larget, B., Liu, L., Suchard, M. A., and Huelsenbeck, J. P. (2012). Mrbayes 3.2: efficient bayesian phylogenetic inference and model choice across a large model space. *Systematic biology*, 61(3):539–542.

Roshan, U., Moret, B. M., Williams, T. L., and Warnow, T. (2004). Rec-i-dcm3: A fast algorithmic technique for reconstructing large phylogenetic trees. In *Proc. 3rd IEEE Computational Systems Bioinformatics Conf. CSB" 04*, LCBB-CONF-2004-002, pages 98–109. IEEE Press.

Sanderson, M. J., McMahon, M. M., Stamatakis, A., Zwickl, D. J., and Steel, M. (2014). Impacts of terraces on phylogenetic inference. *arXiv preprint arXiv:1410.8071*.

Sanderson, M. J., McMahon, M. M., and Steel, M. (2011). Terraces in phylogenetic tree space. *Science*, 333(6041):448–450.

Schmidt, H. A., Strimmer, K., Vingron, M., and von Haeseler, A. (2002). Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3):502–504.

Shimodaira, H. and Hasegawa, M. (2001). Consel: for assessing the confidence of phylogenetic tree selection. *Bioinformatics*, 17(12):1246–1247.

Stamatakis, A. (2005). An efficient program for phylogenetic inference using simulated annealing. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE.

Stamatakis, A. (2011). Phylogenetic search algorithms for maximum likelihood. *Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications*, pages 547–577.

1.2:18 REFERENCES

Stamatakis, A. (2014). Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313.

Stamatakis, A. and Alachiotis, N. (2010). Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data. *Bioinformatics*, 26(12):i132–i139.

Stamatakis, A., Hoover, P., and Rougemont, J. (2008). A rapid bootstrap algorithm for the raxml web servers. *Systematic biology*, 57(5):758–771.

Swofford, D. L. (2001). Paup*: Phylogenetic analysis using parsimony (and other methods) 4.0. b5.

Vos, R. (2003). Accelerated likelihood surface exploration: the likelihood ratchet. *Systematic Biology*, 52(3):368–373.

Zwickl, D. (2006). *GARLI, vers. 0.951. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion.* PhD thesis, Ph. D. dissertation, University of Texas, Austin, Texas, USA.